Anomaly Detection using Autoencoders in High Performance Computing Systems

Andrea Borghesi[1], Andrea Bartolini[1], Michele Lombardi[1], Michela Milano[1], and Luca Benini[1,2]

[1] DISI/DEI, University of Bologna
[2] Integrated System Laboratory, ETHZ

Abstract. Anomaly detection in supercomputers is a very difficult problem due to the big scale of the systems and the high number of components. The current state of the art for automated anomaly detection employs Machine Learning methods or statistical regression models in a supervised fashion, meaning that the detection tool is trained to distinguish among a fixed set of behaviour classes (healthy and unhealthy states).
We propose a novel approach for anomaly detection in High Performance Computing systems based on a Machine (Deep) Learning technique, namely a type of neural network called *autoencoder*. The key idea is to train a set of autoencoders to learn the normal (healthy) behaviour of the supercomputer nodes and, after training, use them to identify abnormal conditions. This is different from previous approaches which where based on learning the abnormal condition, for which there are much smaller datasets (since it is very hard to identify them to begin with).
We test our approach on a real supercomputer equipped with a fine-grained, scalable monitoring infrastructure that can provide large amount of data to characterize the system behaviour. The results are extremely promising: after the training phase to learn the normal system behaviour, our method is capable of detecting anomalies that have never been seen before with a very good accuracy (values ranging between 88% and 96%).

1 Introduction

High Performance Computing (HPC) systems are complex machines with many components that must operate concurrently at the best of their theoretical performance. In reality, many factors can degrade the performance of a HPC system: hardware can break, the applications may enter undesired and unexpected states, components can be wrongly configured. A critical aspect of modern and future supercomputers is the capability of detecting faulty conditions stemming from the improper behaviour of one or multiple parts. This issue is relevant not only for scientific computing systems but also in data centers and clouds providers, whose business strongly relies on the availability of their web services. An automated process for anomaly detection would be a great improvement for current HPC systems, and it will probably be a necessity for future Exascale supercomputers. Nowadays, monitoring infrastructures are available in many HPC systems and data centers, used to gather data about the state of the systems. Given the

Copyright © 2019 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR-WS.org/Vol-2495/paper3.pdf

deluge of data , real-time identification of problems and undesired situations is a daunting task for system administrators. In this paper we present a novel approach to deal with this issue, relying on a fine-grain monitoring framework and on an autonomous anomaly detection method that uses Machine Learning (ML) techniques.

Automated anomaly detection is still a relatively unexplored area in the HPC field. The current state-of-the-art relies on *supervised* [18] ML methods that learn to distinguish between healthy and faulty states after a training phase during which the supercomputer must be subjected to both conditions (*labeled* training data). This requirement complicates the training process: in HPC systems, data is very abundant but labels are scarce. However, in supercomputers the normal behaviour is predominant – and can be deterministically restored by system administrators. The same cannot be said for faulty behaviour, which is undesired, sporadic and uncontrolled. Furthermore, labelling faulty conditions is a expensive task and thus the correct labeled data sets required by typical supervised approaches are not available in supercomputers.

Conversely, there is another type of ML that does not require any label and it is referred to as *unsupervised* [18] learning. In this case the data set contains only the features describing the system state and no labels; the learning algorithm learns useful properties about the structure of the dataset. To address the issue, we propose an anomaly detection method less dependent on labeled data; to be precise, our approach belongs to the *semi-supervised* branch of ML, which combines the two methodologies described before. Our idea is to use autoencoders [11] to learn the normal behaviour of supercomputer nodes and then to use them to detect abnormal states. In our method we require labels during the pre-processing phase because we need to obtain a data set containing only normal conditions. After this "normal" data set has been obtained the training of the ML model proceeds in unsupervised fashion, without the need of labels. A critical advantage of our method is that it will be able to identify faulty conditions even though these have not been encountered earlier during the training phase. With our method we do not need to inject anomalies during the training phase (possibly not feasible in a production system) and we do not require system logs or changes to the standard supercomputer users' work flow.

The main contributions of our approach are: 1) a very precise anomaly detection rate (up to 88%-96% accuracy); 2) identification of new types of anomalies unseen during the initial training phase (thanks to its semi-supervised nature); 3) no need for large amount of labeled data. To demonstrate the feasibility of our approach we consider a real supercomputer hosted by the Italian inter-universities consortium CINECA [1]. We use historical data collected with an integrated monitoring system to train our autoencoders and then we test them by injecting anomalies in a subset of the computing nodes; the experimental results show how this approach can distinguish between normal and anomalous states with a very high level of accuracy.

2 Related Works

Tuncer et al. [19] deal with the problem of diagnosing performance variations in HPC systems. The approach is based on the collection of several measurements gathered by a monitoring infrastructure; from these measures, a set of statistical features describ-

ing the state of the supercomputer is extracted. The authors then train different ML algorithms to classify the behaviour of the supercomputer using the statistical features previously mentioned. Unfortunately the authors propose a supervised approach which is not perfectly suited for the HPC context. Dani et al. [8] present an unsupervised approach for anomaly detection in HPC. Their work is remarkably different from our approach since they do not rely on a monitoring infrastructure but consider only the console logs generated by computing nodes.

Although not yet applied to the HPC field, Deep Learning based approaches for anomaly detection have been studied in other areas [7, 15, 14], especially in recent years. Lv et al. [17] propose a deep learning based algorithm for fault diagnosis in chemical production systems. The proposed method is capable of real time detection and classification and, moreover, it can do the diagnosis online. Nevertheless, their approach is supervised and thus it definitely differs from ours. Lee et al. [16] introduce a convolutional neural network (CNN) model for fault identification and classification in semiconductor manufacturing processes. This method makes it possible to locate the variable and time information that represents process faults. Ince et al. [12] discuss a CNN-based method for electrical motor fault detection; their method can work directly on the raw measurement data, with no preprocessing. The neural network combines feature extraction and classification, but proceeds in a supervised manner.

3 Data Collection

A very important aspect for our anomaly detection approach is the availability of large quantity of data that monitors and thus describes the state of a supercomputer. To test our approach we take advantage of a HPC system with an integrated monitoring infrastructure, D.A.V.I.D.E.[2], an energy efficient supercomputer hosted by CINECA in Bologna, Italy. It has by 45 nodes with a total peak performance of $990\,\mathrm{TFlops}$ and an estimated power consumption of less than $2\,\mathrm{kW}$ per node. The system was ranked #440 in TOP500 [9] and #18 in GREEN500 [10] in November 2017 list. The data collection infrastructure deployed in D.A.V.I.D.E. is called *Examon* and has been presented in previous works [4, 3]. Examon is a fine-grained, lightweight and scalable monitoring infrastructure for Exascale supercomputers. The data coming from heterogeneous data sources is gathered in an integrated and uniform repository, making it very easy to create data sets providing a holistic view of the supercomputer and thus describing the system state. Due to storage limitation, fine-grained data older than a week is discarded but job information and coarse-grained data are preserved long-term. For this paper, we work with the coarse-grained data aggregated in 5-minutes long intervals. Furthermore, we focused on a subset of the data collected by Examon; for each node we have 166 metrics (our *features*), i.e. core loads, temperatures, power consumptions, etc.

4 The Autoencoder-based Approach

We aim at detecting anomalies that happen at the node-level. Currently, we focus on single nodes. We create a set of separate autoencoder models, one for each node in

the system. Each model is trained to learn the normal behaviour of the corresponding node and to be activated if anomalous conditions are measured. If an autoencoder can learn the correlations between the set of measurements (features) that describe the state of a supercomputer, then it can consequently notice changes in these correlations that indicate an abnormal state. Under normal operating conditions these features are linked by specific relations (i.e. the power consumption of a core is directly related to the workload and temperature to the power and frequency). We hypothesize that these correlations will be perturbed if the system enters in an anomalous state.

The *reconstruction error* is the element we use to detect anomalies. An autoencoder can be trained to minimize this error. In doing so, it learns the relationships among the features of the input set. If we feed a trained autoencoder with data not seen during the training phase, it should reproduce the new input with good fidelity, at least if the new data resemble the data used for the training. If this is not the case, the autoencoder cannot correctly reconstruct the input and the error will be greater. We propose to detect anomalies by observing the magnitude of the reconstruction error.

All autoencoders have the same structure. We opted for a fairly simple structure composed by three layers: I) an input layer with as many neurons as the number of features (166), II) a densely connected intermediate sparse layer [5] with 1660 neurons (ten times the number of features) with Rectified Linear Units (*ReLu*) as activation functions and a L1 norm regularizer [11], III) a final dense output layer with 166 neurons with linear activations. This network was obtained after an empirical evaluation, after having experimented with different topologies and parameter configurations. To summarize, our methodology has the following steps: 1) create an autoencoder for each computing node in the supercomputer; 2) train the autoencoders using data collected during normal operating conditions; 3) identify anomalies in new data using the reconstruction error obtained by the autoencoders.

5 Experimental Evaluation

In every HPC system there are multiple possible sources of anomalies and fault conditions, ranging from hardware faults to software errors. In this paper we verify the proposed approach on a type of anomaly that easily arises in real systems and happens at the level of single nodes, namely *misconfiguration*. More precisely, we consider the misconfiguration of the frequency governor of a computing node. Modern Linux systems allow to specify different policies regulating the clock speed of the CPUs, thanks to kernel-level drivers referred as frequency governors [6]. Different policies have different impacts on the clock speed, frequency and power consumption of the CPUs.

We considered three different policies. The first one, *conservative*, is the default policy on D.A.V.I.D.E. (the normal behaviour); it sets the CPU clock depending on the current CPU load. Two other types of policies have been used to generate anomalies, i) the *powersave* policy and ii) the *performance* policy. These frequency governors statically set the CPU to the, respectively, lowest and highest frequency in the allowed range.

5.1 Results

In this work we used an off-line approach. We gathered the measurements collected during months of real usage of D.A.V.I.D.E. and we created a data set; the data is normalized to have values in the range $[0, 1]$. The data set is split in 3 components: 1) the training set D_{Train} (containing data points within periods of normal behaviour), 2) the test set without anomalies D^N_{Test} (again, only periods of normal behaviour) and 3) the test set with anomalies D^A_{Test} (the periods when we injected anomalies on some nodes).

For these experiments we selected a subset of the data collected by Examon during D.A.V.I.D.E. lifetime. The period we considered is 83 days long, from March 2018 to May 2018. During this period D.A.V.I.D.E. was in the normal state for most of the time – 66 days, 80% of the time – while we forced anomalous states for smaller sub-periods of a few days, 13 days in total. Since we know when the anomalies were injected identifying D^A_{Test} is trivial. D_{Train} and D^N_{Test} were created by randomly splitting the data points belonging to the 66 days of normal state, 80% of the data points going to D_{Train} and 20% to D^N_{Test}.

Each autoencoder is trained with *Adam* [13] optimizer with standard parameters, minimizing the mean absolute error; the number of epochs used in the training phase is 100 and the batch size has a fixed value (32). These values were chosen after a preliminary exploration because they guarantee very good results with very low computational costs. The time required to train the network is around 5 minutes on a quad-core processor (Intel i7-5500U CPU 2.40GHz) with 16GB of RAM (without using GPUs).

Reconstruction Error-Based Detection As explained previously, our anomaly detection method relies on the hypothesis that an autoencoder can be taught to learn the correlations among the features in a data set representing the healthy state of a supercomputer node. In this case the autoencoder would be capable to reconstruct an input data set never seen before, if this new input resembles the healthy one used during the training phase – if in the unseen data set the features correlations are preserved. Conversely, an autoencoder would struggle to reconstruct data sets where the learned correlations do not hold. To demonstrate our hypothesis, we expect to observe higher reconstruction errors for the anomalous periods with respect to the error obtained in normal periods. We are not strictly interested in the absolute value of the reconstruction error but rather on the relative difference between normal and anomalous periods.

This reconstruction error is plotted in Figure 1; it displays the results computed for node *davide45* (other nodes were omitted for space reason but their behaviour is very similar). The x-axis and y-axis show, respectively, the time and the normalized reconstruction error (we sum the error for each feature and divide by the number of features NF). The reconstruction error trend is plotted with a light blue line; the gaps in the line represent periods when the node was idle and that have been removed from the data set.

We observe 6 anomalous periods (highlighted by colored lines along the x-axis): during the first 5 (red lines) the frequency governor was set to powersave while during the last one (blue) the governor was set to performance. The reconstruction error

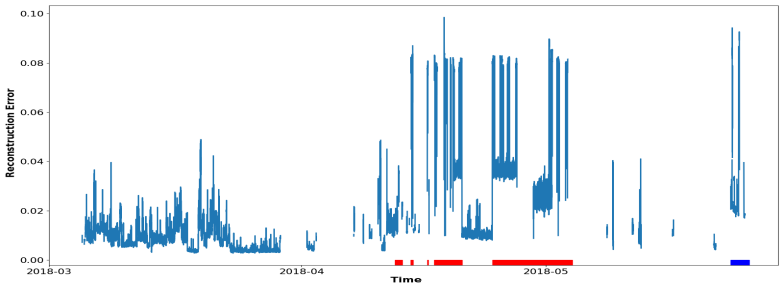
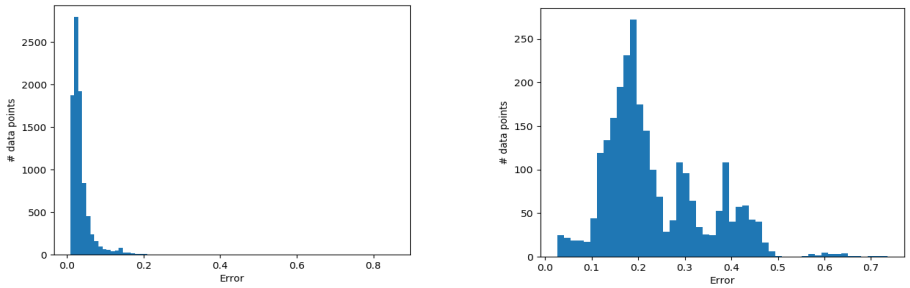


Fig. 1: Reconstruction error for node *davide45*

is never exactly zero, but this is not our concern: our analysis does not rely on the absolute value of the error, but rather on the relative magnitude of the errors computed for different data sets. The reconstruction error is indeed greater when the nodes are in an anomalous state, as underlined by the higher values in the y-axis in the periods corresponding to anomalies. Hence, the autoencoder struggles to recreate the "faulty" input data set. Although the plot shown is promising, it does not actually show that the reconstruction error for unseen healthy input is actually lower than the error committed with anomalous periods. This happens because the normal behaviour data set was randomly split in the subset D_{Train} and D^N_{Test} and it is impossible to distinguish between them by simply looking at the plot. However, our insight is backed by the quantitative analysis. To measure the quality of the anomaly detection we rely on the Mean Absolute Error (MAE) and on the Root Mean Squared Error (RSME). For each autoencoder we computed MAE and RSME for every set D_{Train}, D^N_{Test} and D^A_{Test}.

The results obtained for all autoencoders are very similar but in order to make a fair comparison between different nodes we do not use the absolute values of MAE and RSME but we rather employ a normalized version: the normalized MAE (RSME) is obtained by dividing the actual MAE (RSME) by the MAE (RSME) computed for D_{Train}. In this way we force the normalized error for the training set to be equal to 1 (since we are not strictly interested in its absolute value) and we highlight the relative difference of error between sets. If the normalized error for a test set is close to one this means that the autoencoder was able to reconstruct the input quite well; larger errors imply that the autoencoder was not capable to reproduce the input – these situations are those that we claim to be anomalies. The normalized MAE for D^N_{Test} is equal to 1.08 and the MAE for D^A_{Test} is 14.54; the normalized RMSE are, respectively, 1.17 and 11.18. The errors for D_{Train} are always equal to 1 (due to the normalization).

The results clearly indicate that our hypothesis holds true (as hinted also by the previous plot with the reconstruction error). Both the average normalized MAE and RSME for the test set with no anomalies D^N_{Test} are very close to 1, suggesting that the autoencoders have correctly learned the correlations between the measured features of a healthy system. Therefore, when the autoencoders are fed with unseen input that preserve these correlations they can reconstruct it with good precision. On the contrary, the autoencoders cannot correctly reproduce new input that does not resemble a healthy



(a) Normal data set (b) Anomaly data set

Fig. 2: Error distribution for node *davide45*

system, that is a system in an anomalous state. This is shown by the markedly higher normalized MAE and RSME obtained for D^A_{Test}.

Detection Accuracy So far we have observed the reconstruction error trends obtained by our approach based on autoencoders, but we still have to discuss how the reconstruction error can be used to actually detect an anomaly. Our goal is to identify an error threshold θ to discriminate between normal and anomalous behaviour. In order to do so we shall start by looking at the distributions of the reconstruction errors. Again, we are considering each autoencoder (and thus corresponding node) separately. We distinguish the errors distribution for healthy data sets ($D_{Train} \cup D^N_{Test}$) and for the unhealthy data set (D^A_{Test}). Figure 2 shows the error distributions for the autoencoder corresponding to node *davide45* – again other nodes have the same behaviour. The graph contains the histograms of the error distributions; in the x-axis we have the reconstruction error and in the y-axis there is the number of data points with the corresponding error. The left-most sub-figure (Fig. 2a) shows the error distribution for the normal data set ($D_{Train} \cup D^N_{Test}$) and the other one (Fig. 2b) shows the distribution for the anomalous data set. It is quite easy to see that the errors distribution of the normal data set is extremely different from the anomalous one.

Since we can clearly distinguish the error distributions we opted for a simple method to classify each data point: if the reconstruction error E_i for data point i is greater than a threshold θ, then the point is "abnormal"; otherwise the data point is considered normal. The next step is to identify the threshold used to classify each data point. We choose as a threshold the n-th percentile of the errors distribution of the normal data set, where n is a value that depends on the specific autoencoder/node. In order to find the best n value for each autoencoder we employed a simple generate-and-test search strategy, that is we performed experiments with a finite number of values and then chose those guaranteeing the best results in term of classification accuracy. Generally, the best results are obtained with higher thresholds, i.e. $n \geq 93$. To asses the accuracy of the classification we compute the *F-score* for each class, *normal* (N) and *anomaly* (A). In Table 1 we see some results. In the first column from the left there is the node whose autoencoder

F-score values are reported (we report the values for only a subgroup of nodes). The remaining columns report the F-score values for 3 different n-th percentiles (and therefore different thresholds); there are two F-score values for each n-th percentile, one computed for the normal class (*N*) and one for the anomaly class (*A*).

Node	95-th perc.		97-th perc.		99-th perc.	
	N	A	N	A	N	A
davide17	0.97	0.89	0.98	0.93	0.99	0.97
davide19	0.97	0.90	0.98	0.94	0.99	0.97
davide45	0.97	0.92	0.98	0.95	0.99	0.98
davide27	0.95	0.90	0.91	0.77	0.86	0.52
davide28	0.94	0.88	0.96	0.89	0.90	0.69
davide29	0.97	0.75	0.98	0.82	0.99	0.85
Average	0.96	0.87	0.96	0.88	0.95	0.82

Table 1: Classification Results

The table can be divided in three subparts (separated by horizontal lines): 1) the first one contains nodes similar to *davide45*, i.e. nodes where most of the anomalies were of type powersave; 2) the second group is comprised of nodes where most of the anomalies had the frequency governor set to performance; 3) the last group (the last row) is the average of the other nodes. In general we can see that the F-score values are very good, highlighting the high accuracy of our approach. A notable difference can be observed between the two sub-groups of nodes. In nodes with a prevalence of powersave anomalies higher thresholds (higher n-th values) guarantee better results: this happens because, as seen for instance in Figure 2, the error distributions are more separable. In the case of nodes characterized by more anomalies of performance type, increasing the threshold does not necessarily improve the accuracy – although this can still occur for some nodes. In these nodes it is harder to distinguish normal data points from anomalies of type performance (since they behave similarly).

6 Conclusion

In this paper we proposed an approach to detect anomalies in a HPC system that relies on large data sets collected via a lightweight and scalable monitoring framework and employs autoencoders to distinguish between normal and anomalous system states.

In the future we plan to further validate our method by testing it on a broader set of anomalies. Our goal is to expand the anomaly detection technique in order to be able to also classify different types of anomalies; in addition to recognize that the system is in an anomalous state, the autoencoder (possibly a refined and more complex version) will be also able to distinguish among different anomaly classes and sources. We also plan to implement our approach in a on-line prototype to perform real-time anomalous detection on a supercomputer, again using D.A.V.I.D.E. as a test bed.

References

1. Cineca inter-university consortium web site. http://www.cineca.it//en, accessed: 2018-06-29
2. Ahmad, W.A., Bartolini, A., Beneventi, F., et al.: Design of an energy aware petaflops class high performance cluster based on power architecture. In: 2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW) (May 2017). https://doi.org/10.1109/IPDPSW.2017.22
3. Bartolini, A., Borghesi, A., Libri, A., et al.: The D.A.V.I.D.E. big-data-powered fine-grain power and performance monitoring support. In: Proceedings of the 15th ACM International Conference on Computing Frontiers, CF 2018. pp. 303–308 (2018). https://doi.org/10.1145/3203217.3205863, http://doi.acm.org/10.1145/3203217.3205863
4. Beneventi, F., Bartolini, A., Cavazzoni, C., et al.: Continuous learning of hpc infrastructure models using big data analytics and in-memory processing tools. In: Proceedings of the Conference on Design, Automation & Test in Europe. European Design and Automation Association (2017)
5. Boureau, Y.l., Cun, Y.L., et al.: Sparse feature learning for deep belief networks. In: Advances in neural information processing systems. pp. 1185–1192 (2008)
6. Brodowski, D., Golde, N.: Cpu frequency and voltage scaling code in the linux (tm) kernel. Linux kernel documentation (2013)
7. Costa, B.S.J., Angelov, P.P., Guedes, L.A.: Fully unsupervised fault detection and identification based on recursive density estimation and self-evolving cloud-based classifier. Neurocomputing **150**, 289–303 (2015)
8. Dani, M.C., Doreau, H., Alt, S.: K-means application for anomaly detection and log classification in hpc. In: International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems. pp. 201–210. Springer (2017)
9. Dongarra, J.J., Meuer, H.W., Strohmaier, E.: 29th top500 Supercomputer Sites. Tech. rep., Top500.org (Nov 1994)
10. Feng, W.c., Cameron, K.: The green500 list: Encouraging sustainable supercomputing. IEEE Computer **40**(12) (December 2007)
11. Goodfellow, I., Bengio, Y., Courville, A., Bengio, Y.: Deep learning, vol. 1. MIT press Cambridge (2016)
12. Ince, T., Kiranyaz, S., Eren, L., et al: Real-time motor fault detection by 1-d convolutional neural networks. IEEE Transactions on Industrial Electronics (2016)
13. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
14. Kiran, B.R., Thomas, D.M., Parakkal, R.: An overview of deep learning based methods for unsupervised and semi-supervised anomaly detection in videos. Journal of Imaging (2018)
15. Kwon, D., Kim, H., Kim, J., et al.: A survey of deep learning-based network anomaly detection. Cluster Computing (2017)
16. Lee, K.B., Cheon, S., Kim, C.O.: A convolutional neural network for fault classification and diagnosis in semiconductor manufacturing processes. IEEE Transactions on Semiconductor Manufacturing **30**(2), 135–142 (2017)
17. Lv, F., Wen, C., Bao, Z., Liu, M.: Fault diagnosis based on deep learning. In: American Control Conference, 2016. IEEE (2016)
18. Mitchell, T.M.: Machine learning and data mining. Communications of the ACM **42**(11), 30–36 (1999)
19. Tuncer, O., Ates, E., Zhang, Y., et al.: Diagnosing performance variations in hpc applications using machine learning. In: International Supercomputing Conference. pp. 355–373. Springer (2017)