A Semisupervised Autoencoder-based Approach for Anomaly Detection in High Performance Computing Systems

Andrea Borghesi[a], Andrea Bartolini[b], Michele Lombardi[a], Michela Milano[a], Luca Benini[b,c]

[a] *DISI, University of Bologna. Viale Risorgimento 2, 40123, Bologna, Italy*
[b] *DEI, University of Bologna. Viale Risorgimento 2, 40123, Bologna, Italy*
[c] *Integrated Systems Laboratory at ETH Zurich, Switzerland*

Abstract

High Performance Computing (HPC) systems are complex machines with heterogeneous components that can break or malfunction. Automated anomaly detection in these systems is a challenging and critical task, as HPC systems are expected to work 24/7. The majority of the current state-of-the-art methods dealing with this problem are Machine Learning techniques or statistical models that rely on a supervised approach, namely the detection mechanism is trained to recognize a fixed number of different states (i.e. normal and anomalous conditions).

~~We propose a semi-supervised approach for anomaly detection in supercomputers, based on a type of neural network called *autoencoder.*~~ **In this paper a novel semi-supervised approach for anomaly detection in supercomputers is proposed, based on a type of neural network called *autoencoder*.** ~~Our~~**The** approach learns the normal state of the supercomputer nodes and after the training phase can be used to discern anomalous conditions from normal behaviour; in doing so it relies only on the availability of data characterizing only the normal state of the system. This is different from supervised methods that require datasets with many examples of anomalous states, which are in general very rare and/or hard to obtain.

~~We tested our approach on a real-life High Performance Computing system equipped with a monitoring infrastructure capable to generate large amount of data describing the system state.~~ **The approach was tested on a real-life High Performance Computing system equipped with a monitoring infrastructure capable to generate large amount of data describing the system state.** ~~Our approach~~**The proposed approach** definitely outperforms the best current techniques for semi-supervised anomaly detection, with an increase in accuracy detection of around 12%. ~~We also present two different implementations of our approach, one where each~~ **Two different implementations are discussed: one where each** supercomputer node has a specific model and one with a single, generalized model for all nodes, in order to explore the trade-off between accuracy and ease of deployment.

Keywords: Anomaly Detection, High Performance Computing, Autoencoder, Machine Learning

1. Introduction

Supercomputers are large systems comprising multiple high-performance subsystems working concurrently and close to the peak of their optimal theoretical capability. Actually, many things could go wrong and cause a decrease in the overall performance of a High Performance Computing (HPC) machine: hardware can break, malfunction or be incorrectly configured, software programs can contain bugs or reach undesirable states. Hence, ensuring reliability and availability is a major issue in the HPC context. A key

Email addresses: andrea.borghesi3@unibo.it (Andrea Borghesi), a.bartolini@unibo.it (Andrea Bartolini), michele.lombardi2@unibo.it (Michele Lombardi), michela.milano@unibo.it (Michela Milano), luca.benini@unibo.it,luca.benini@iis.ee.ethz.ch (Luca Benini)

challenge to be addressed by researchers and practitioners is the detection of anomalies and fault conditions that can arise due to the incorrect or sub-optimal behaviour of a wide variety of components. This is an important problem for both the scientific computing community and data centers and clouds providers, since their business strongly depends on the 24/7 availability of their services. For example, it was estimated that in 2016 Amazon would have lost 15M$ for one hour of downtime[1]. This strongly motivates the requirement of an automated procedure for anomaly detection in current supercomputers and data centers, and this need will become even more pressing for future Exascale systems[2].

Luckily, many modern HPC machines and data centers are equipped with a monitoring infrastructure that collects information characterizing the state of the system and underlying components, relying on a vast array of measuring sensors. ~~This large amount of data can be exploited to cope with the task of identifying faulty conditions.~~ However, the large amount of data available is both a boon and a challenge, since a real-time analysis of the information flow is a daunting task for system administrators. ~~With the purpose of mitigating this problem, in this paper we present an automated approach for anomaly detection in HPC systems that makes use of the data collected by a fine-grain monitoring framework and relies on Machine Learning (ML) techniques.~~ **To mitigate this problem, in this paper an automated approach for anomaly detection in HPC systems is presented; it makes use of the data collected by a fine-grain monitoring framework and relies on Machine Learning (ML) techniques.**

The current state-of-the-art methods for anomaly detection in supercomputers belong to the *supervised*[3] Machine Learning class; these are techniques that discern normal and faulty states after having been "taught" to do so during a training phase. In the training phase both normal and anomalous examples must be provided in order to assure the success of the learning task ~~; in the ML language, a *labeled* data set is needed~~. **a *labeled* data set is needed**. ~~A supervised algorithm goal is then to learn the relation between a certain set of input features and the corresponding label (normal state and possibly multiple anomaly classes).~~ To guarantee a successful learning process the training data set should be unbiased and balanced[4], that is containing more or less the same number of examples for each class. ~~This requirement in conjunction with the need of labels greatly complicates the training task: in supercomputers, large amount of data is available but labels are lacking. This is originated by the difficulty of identifying anomalies in the first place (moreover, the definition of "anomaly" itself can vary in different situations).~~ **These requirements greatly complicate the training task: in supercomputers, large amount of data is available but labels are scarce**[1]**.** ~~In addition, even when detected, fault conditions are not always stored in logs or databases, but rather the task of assigning labels to a data set – identifying the corresponding classes as healthy or unhealthy – is the sole responsibility of system administrators.~~ **In addition, fault conditions are not always stored in logs or databases, but rather the task of assigning labels to a data set is the sole responsibility of system administrators.** Hence, it is not always possible to have the correct labeled data sets needed by supervised approaches. This is a widely acknowledged problem and some authors have proposed in recent years to create balanced labeled data sets using fault injections tools (for instance Netti et al.[5]); these approaches are promising, however none has been actually deployed and tested on a real in-production supercomputer.

~~To help dealing with this problem, a different ML branch can provide assistance since it does not require labels and is referred to as *unsupervised*[3, 6] learning. With this methodology the learning algorithm is fed exclusively with a data set containing the features describing the system state (without labels); during the training the algorithm learns useful properties about the structure of the dataset. Unfortunately, unsupervised learning is a very complex task and obtaining accurate anomaly detection with this scheme is very difficult.~~ **Another ML branch does not require labels and is referred to as *unsupervised*[3, 6] learning. This methodology requires only a data set containing the features describing the system state (without labels); during the training the algorithm learns useful properties about the structure of the dataset. Unfortunately, unsupervised learning is very complex and obtaining accurate anomaly detection with this scheme is very difficult.** ~~Therefore, in previous work[7] we proposed an approach belonging to a third branch of ML, namely *semi-supervised* learning, that~~

[1]Identifying anomalies is difficult in the first place and the definition of "anomaly" itself can vary in different situations.

~~uses labeled data but with looser requirements than supervised learning.~~ **Therefore, previous works[7] proposed an approach belonging to a third branch of ML, namely *semi-supervised* learning, that uses partially labeled data.** ~~More precisely, we rely on *autoencoders*[8] (a type of neural network) to learn the normal state of a computing node of a HPC system (thus we use a data set containing only data describing normal behaviour); once trained, the autoencoders can be used to identify anomalous situations. We need labels just to obtain the normal data set; afterwards, the training proceeds in a unsupervised fashion. With our method we can detect faults that were not included in the training data set and we do not need special system logs or changes to the typical users' work flow.~~ **This method relies on *autoencoders*[8] to learn the normal state of the computing nodes of a HPC system (the training set contains only data describing normal behaviour); once trained, the autoencoders can identify anomalous situations. The labels are needed just to obtain the normal data set; afterwards, the training proceeds in an unsupervised fashion. In this way, faults not included in the training set can be detected too.** Moreover, the proposed approach does not need to inject anomalies during the training phase (whereas supervised techniques have this requirement), an action that cannot be possible in the majority of production HPC systems without incurring in serious downtime.

~~In this paper we significantly extend our previous work by evaluating in depth~~ **This paper significantly extends the previous work by evaluating in depth** the autoencoder-based approach from an engineering and practical point of view and comparing it with the state-of-the art. The main contributions of this paper are the following:

- a comprehensive experimental analysis of our approach through the evaluation of a larger set of computing nodes belonging to a HPC cluster;
- comparison of two different models based on autoencoders, I) a node-specific model and II) a single general model, evaluating their performance and deployment challenges;
- a thorough comparison with alternative techniques from the semi-supervised field, revealing that our approach has an overall detection accuracy around 12% higher than other methods – we also compare our approach with the current state-of-the-art method for anomaly detection in HPC, which is based on supervised ML, hence not directly comparable to ours, and we show that the gap is very small (around 5%);
- practical guidelines to implement the proposed approach, namely the requirement in terms of data to be collected in order to train the models and trade-off between data set size and detection accuracy.

~~To demonstrate the effectiveness of our approach we deployed it on a real HPC system, D.A.V.I.D.E., hosted by the Italian inter-universities consortium CINECA[9].~~ **The approach has been deployed on a real HPC system, D.A.V.I.D.E., hosted by the Italian inter-universities consortium CINECA[9].** A monitoring infrastructure for large-scale data collection and storage has been installed on this supercomputer, ~~providing us with the ideal testbed to create the data set needed to train our autoencoder-based technique and to assess its accuracy using real data.~~ **ideally suited to create the training set needed for the autoencoder-based technique and to assess its accuracy using real data.** The paper has the following structure. ~~In Section 2 we discuss the state-of-the-art for anomaly detection in HPC systems (supervised techniques) and other approaches from the literature belonging to the unsupervised and semi-supervised areas but never applied to supercomputers. In Section 3 we introduce the target system (D.A.V.I.D.E.) and describe its monitoring infrastructure. Section 4 then introduces the autoencoder-based approach, providing a detailed explanation of its mode of operation. Afterwards, Section 5 describes exhaustively the experimental evaluation performed to test the accuracy of our approach; it contains the method used to inject anomalies in the HPC cluster, the comparison with different types of models and the comparison with the state-of-the-art. Finally, Section 6 summarizes the work presented previously and concludes the paper.~~ **Section 2 discusses the state-of-the-art for anomaly detection in HPC systems (supervised techniques) and other approaches from the literature belonging to the unsupervised and semi-supervised areas but never applied to supercomputers. Section 3 describes the target system (D.A.V.I.D.E.) and its monitoring infrastructure. Section 4 then**

introduces the autoencoder-based approach, providing a detailed explanation of its mode of operation. Section 5 reports the results of the experimental evaluation; it contains the method used to inject anomalies in the HPC cluster, the comparison with different types of models and the comparison with the state-of-the-art. Finally, Section 6 summarizes the work and concludes the paper.

2. Related Works

Tuncer et al.[10, 11] propose a method to diagnose performance variations in HPC systems; performance variation can be also used to describe the difference between normal and anomalous behaviours. They use a large set of different measures collected by a monitoring framework; from these measurements, they extract several statistical features describing the state of the HPC system. These features then compose a data set used to train several ML algorithms in order to classify the system behaviour; the best results are obtained with a Random Forest classifier[12]. Their results are very good and outperform many previous methods that followed similar schemes (i.e. [13, 14]). This approach is the current state-of-the-art for anomaly detection in HPC systems, to the best of our knowledge.

Baseman et al.[15] discuss an analogous technique for fault detection in supercomputers. They apply a statistical method called *classifier-adjusted density estimation* (CADE) to the HPC context. The main idea of CADE is to combine a uniform density estimate and the probabilistic output of a classifier in order to obtain an accurate density estimator, which can be used to discern anomalous states from normal ones. Their approach begins by extracting temporal relational features and their gradients from the sensor data (coming from a supercomputer). Then they use both real and artificially generated data (thanks to density estimation) to train a supervised classifier (a Random Forest) to classify each data point depending on its "anomalousness"; the output of the classifier is not a single class label but rather a probability.

The methods just discussed share a couple of important elements to consider. ~~First, the authors employ supervised ML techniques and this means that the training set must contain examples of all classes to be detected, namely healthy and unhealthy states. This implies that, to properly train the classifiers, a first phase is needed to create a labeled data set, when the supercomputer must run in each of the target faulty conditions.~~ **First, the authors employ supervised ML techniques; hence a first phase is needed to create a labeled data set, when the supercomputer must run in each of the target faulty conditions.** Secondly, with the supervised approach, the classifier can only learn to identify the classes it has been taught to; a new anomaly encountered at run time and never seen before cannot be properly detected by this approach. ~~Our method, thanks to the semi-supervised learning, overcomes both these limitations; namely our approach does not require labeled and balanced data sets containing different anomaly classes – a strong obstacle in supercomputing systems due the 24/7 availability requirement. In our approach we do not aim only at the accuracy of the anomaly detection, but we rather propose a solution that is sound and feasible from the engineering and deployment point of view.~~ **On the contrary, the method proposed in this paper overcomes both these limitations; labeled and balanced data sets are not needed – a strong obstacle in supercomputers due the 24/7 availability requirement.**

Leaving the supervised learning area, Dani et al.[16] present an unsupervised approach for anomaly detection in HPC. Their approach markedly differs from ours since they do not employ datasets containing information describing the supercomputer status. Their goal is to distinguish log messages generated by faulty events from normal messages created by nodes operating under normal condition. Their method is completely unsupervised and based on clustering methods (k-means). The focus of the work is exclusively on anomalies recognizable by the node itself, since it has to generate the warning messages used to identify faults. ~~In our approach, we do not require additional mechanisms to detect anomalies on the nodes and we rely only on the physical and architectural data gathered by a collection framework.~~ **The proposed approach does not require additional mechanisms to detect anomalies on the nodes since it relies only on the physical data gathered by a collection framework.**

Gabel et al. [17] tackle the problem of predicting machine failures in data centers. Their semi-supervised approach learns the correct behaviour of the machines in a data center through the observation of system performance counters and by learning a statistical model corresponding to the normal state; the trained

model can then identify machines in anomalous states because their behaviours differ from the "healthy" machines, after having applied the statistical tests. They focus on predicting latent faults, errors that are not the result of abrupt changes but rather the product of a long period of degraded performance; this focus limits the scope of their approach, since not all anomalies are the outcome of latent faults, i.e. power outages or unexpected configuration changes. Their method is not applicable to HPC systems since it is based on observing differences among machines in a large scale (and possibly distributed) data center, while a typical supercomputer is composed by a single machine (albeit a large one).

In recent years several works from the area of statistical learning (such as probabilistic models, Bayesian networks, etc.) have been suggested to deal with the issue of fault identification and diagnosis, in a wide range of fields, from heat pumps [18] to electrical motors [19], and also applied to complex systems [20, 21]. These methods are promising but they require labeled data as well and also the knowledge of domain experts.

Fully supervised approaches for anomaly detection based on neural networks have been widely discussed in recent years, although never applied to supercomputers. Wang et al.[22] propose an approach for fault diagnosis on power systems based on sparse stacked autoencoder (SSAE) neural networks. SSAE are are composed by a set of sparse autoencoders disposed in a chain-like structure, where the output of the previous autoencoder is fed as input to the following one. In order to perform the fault classification the final layer of autoencoders is typically a softmax layer. In this approach, Wang et al. use instead a more complex solution: a Principal Component Analysis (PCA) layer to further extract the most significant features and a Support Vector Machine (SVM) with a Gaussian kernel. The accuracy of the classification is around 90%. Shen et al.[23] discuss a similar supervised method for fault diagnosis in rotating machinery employing stacked contractive autoencoders (SCAE). The deep contractive autoencoder is used to extract important features that are then fed into a softmax classification layer; the advantage of SCAE derives from the fact that this type of neural network is invariant to small changes of the inputs due to its penalty term, without requiring any prior knowledge or human (domain expert) interactions. Siegel et al. [24] propose a supervised approach for arc-fault detection in electronic circuits for the Internet-of-Things, based on a deep neural network acting as a classifier and trained on real data. The experimental results are very promising, reaching an accuracy higher that 99% in the binary classification task (the goal is to distinguish between faults and normal state). These three methods based on supervised neural networks [22, 23, 24] have never been applied to the HPC context and have a detection accuracy comparable to the HPC state-of-the art [11] (higher than 95%), but are more computationally demanding~~, hence in this paper we opted to compare our approach only with the latter work, since the reference baseline in terms of detection accuracy is the same~~.

Although not yet applied to the HPC field, many semi-supervised and unsupervised approaches (especially Machine and Deep Learning based) have been proposed to tackle the anomaly detection issue in other fields [25, 26, 27, 28]. In particular DL models have been fostered by the rise in computational power guaranteed by GPUs and the availability of large amount of data to train deep neural networks. Goldstein et al. [29] perform an extensive study comparing the performance of several unsupervised anomaly detection techniques. It is a very thorough work that considers the accuracy of each method together with computational demands and sensitivity to parameters.

3. Target HPC System & Data Collection

~~Our approach depends strongly on the availability of monitoring data from which we extract a description of the HPC system state. The effectiveness of every ML technique relies on the quality of the data set used during the training phase: we then chose a supercomputer with an integrated monitoring framework with data coming from many different sources as a target to deploy and test our approach. In this section we briefly describe the HPC system itself and the data collection infrastructure.~~ **The proposed approach depends on the availability of monitoring data describing the HPC system state, thus the test case had to be a supercomputer with an integrated monitoring framework with data from many different sources. This section describes the chosen HPC system and its data collection infrastructure.**

3.1. D.A.V.I.D.E.

D.A.V.I.D.E. (Development for an Added Value Infrastructure Designed in Europe) [30] is an Energy Aware Petaflops Class HPC system based on Power Architecture and coupled with NVIDIA Tesla Pascal GPUs with NVLink, hosted by CINECA in Bologna, Italy. The design of D.A.V.I.D.E. has been developed by E4 Computer Engineering [31] for PRACE [32], with the goal of obtaining an energy efficient cluster for scientific computing. D.A.V.I.D.E. is based on OpenPOWER platform, a key feature that allowed the out-of-band monitoring used in the monitoring infrastructure. D.A.V.I.D.E. is composed by 45 nodes connected with Infiniband EDR 100 GB/s network, with a total peak performance of 990 TFlops and an estimated power consumption of less than 2 kW per node. Each node hosts two IBM POWER8 Processors with NVIDIA NVLink and four Tesla P100 data center GPUs, with the intra-node communication layout optimized for best performance. The system was ranked #440 in TOP500[33] and #18 in GREEN500[34] in November 2017 list.

3.2. EXAMON Monitoring Framework

The data collection infrastructure deployed in D.A.V.I.D.E. is called *Examon* [35, 36]~~; it was designed to be a scalable monitoring framework for future Exascale supercomputers. One of its important features is that the~~. Data originated by heterogeneous sources (for example from both physical sensors and software modules) is gathered and stored in a single structure ~~(although replicated on multiple physical nodes to guarantee robustness)~~ with a uniform access format. This greatly helps the creation of a data set providing an overall view of the whole cluster and its state.

The primary components of the Examon infrastructure are several low-overhead software daemons running along the computing nodes, tightly coupled with them. These daemons monitor many performance and utilization metrics and the power consumption of each node. The data is then sent to a management backbone, through a lightweight TCP/IP communication layer based on the open-source MQTT (MQ Telemetry Transport) protocol [37]. The power consumption at the plug is measured by an embedded monitoring board (Beaglebone Black, BBB [38]), which samples, pre-processes and sends data via MQTT[39]. The embedded monitoring board periodically issues IBM Amester commands [40], for an out-of-band monitoring of the nodes performance. Amester (Automated Measurement of Systems for Temperature and Energy Reporting) is a software tool to remotely collect power, thermal, and performance metrics from IBM servers. It connects to OpenPOWER systems by accessing the service processor firmware, therefore it does not use any of the processing cycles of the processor and has no impact on performance.

The Amester commands exploit the IPMI interface to the OpenPOWER POWER8 on-chip controller[41, 42] (OCC), to get OCC sensor readings. The IPMI interface (Intelligent Platform Management Interface) [43, 44] provides management and monitoring capabilities independently of the host system's CPU, firmware and operating system. It defines a series of interfaces that can be used for out-of-band monitoring and management of computer nodes. The IPMI Amester commands are sent to the OCC through the board management controller (BMC), using a python script. The python script executes on the embedded monitoring board (BBB). The received data are then sent to the MQTT backbone to be processed by Examon. The granularity of the data in Examon is 5s and 10s respectively for IPMI metrics and OCC metrics.

The final destination of the data collected by the measuring agents is a distributed and scalable time series database (KairosDB[45]), built on top of a NoSQL database Apache Cassandra[46] (the single point of access for all data). ~~Examon also gathers information about the jobs submitted on the supercomputer (thanks to an extension to the job dispatching system).~~ Beside the fine-grained measurements coming from the physical sensors, coarse-grained data is computed and stored **(averaging the fine-grain measures over 5 minutes period)**. ~~In particular for each fine-grain metric we have a corresponding aggregated version composed by the average values computed in a 5 minutes period. A set of python scripts are charged with the data-aggregation task and operate with a very low (negligible) overhead; these scripts run on the same physical nodes deputed to run the Cassandra database.~~

Figure 1 summarizes the scheme of the data gathering framework installed on D.A.V.I.D.E., *Examon*; the figure displays also the anomaly detection scheme (yellow rectangles and numbered circles) that will be described in Sec. 4. The monitoring infrastructure resides on one of the two front-end nodes of the

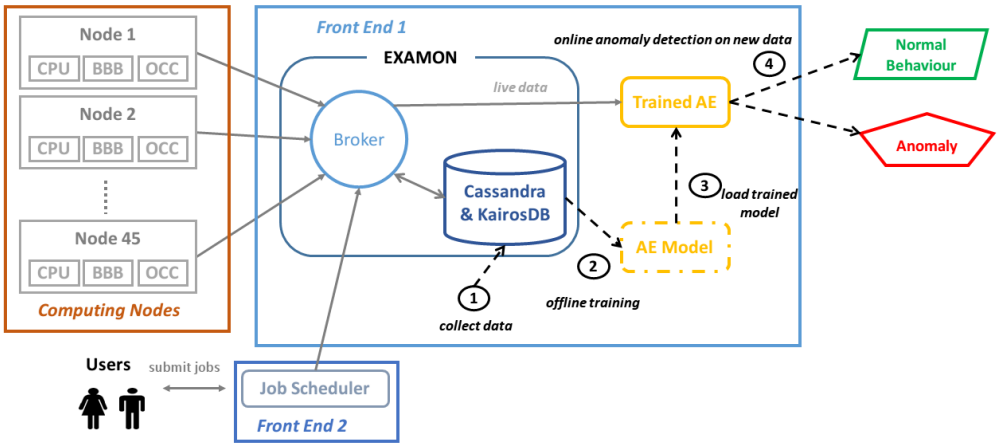


Figure 1: Architecture of data collection infrastructure deployed on D.A.V.I.D.E., plus anomaly detection scheme

supercomputer (*Davide Front-End 1*); the second front-end node hosts the job manager that handles the jobs submitted by users (*SLURM*). The data collected on the computing nodes is gathered by the MQTT central broker; the Kairos and Cassandra databases reside on the same node and are tightly coupled with the broker. Data coming from the 45 computing nodes and the job scheduler is sent to the broker and stored for analysis and visualization purposes.

~~Given the limitations on the storage space available for the data collection framework, it is impossible to store the raw data, due to its sheer amount. The solution was not to save the fine-grained data for longer than a week (short-term storage, with an average occupation around 800GB) and to preserve indefinitely job information and coarse-grained data (long term storage, around 15GB after 13 months of activity). This is a very important fact since in this paper we consider only the coarse-grained, aggregated data.~~ **The approach proposed in this paper considers only coarse-grained, aggregated data, with all the entailed limitations.** For instance this choice clearly limits the type of anomalies that can be detected: events that last for periods shorter than the aggregation time window (5 minutes), and that leave no trace or permanent damage, will not be taken into consideration. The aggregated information then constitutes the data set used to train the autoencoders employed by our approach; for each one of the 45 nodes of D.A.V.I.D.E. there are around 170 metrics (also referred to as the *features*, adopting ML terminology), i.e. core loads, temperatures, fan speed, power consumption of whole nodes, consumption of single subcomponents, etc. Since there are hundreds of metrics, reporting all of them would hinder the comprehension of the paper; this online repository contains a comprehensive description of the Examon infrastructure [47]; the collected metrics can be found in the document called "Metrics List & Description". [2].

4. Anomaly Detection with Autoencoders

~~In this section we describe the autoencoder-based model. The core goal of the model is to learn the "correct" behaviour of a supercomputer, in order to detect anomalous conditions. This approach has already been described in [7]; this section illustrates the key points of that approach, novel contributions will be~~

[2]For reference, the list of Amester sensors names are [48]: PWR250US, PWR250USFAN, PWR250USIO, PWR250USSTORE, PWR250USGPU, PWRAPSSCH0, FREQA2MSP0, IPS2MSP0, PWR250USP0, PWR250USVDD0, CUR250USVDD0, PWR250USVCS0, PWR250USMEM0, SLEEPCNT2MSP0, WINKCNT2MSP0, TEMP2MSP0, TEMP2MSP0PEAK, UTIL2MSP0, FREQ250USP0Cy, FREQA2MSP0Cy, IPS2MSP0Cy, NOTBZE2MSP0Cy, NOTFIN2MSP0Cy, TEMP2MSP0Cy, UTIL2MSP0Cy, NUTIL3SP0Cy, CMBW2MSP0Cy, PWRPX250USP0C, VOLT250USP0VX, MRD2MSP0Mx, MWR2MSP0Mx, M4RD2MSPxMy, M4WR2MSPxMy

~~described in the rest of the paper. We concentrate on detecting anomalies that happen at the node-level.~~ **This section introduces the autoencoder-based model. The core goal of the model is to learn the "correct" behaviour of a supercomputer, in order to detect anomalous conditions. More precisely, the proposed approach focuses on detecting anomalies that happen at the node-level.** ~~In our approach we make a key assumption: an autoencoder can learn the representation of a set of measurements (features) describing the normal state of a supercomputer node (for instance, the power consumption of a core is directly related to the workload); after having learnt them, the model can notice representation changes that underlie anomalous conditions. If an autoencoder is capable to learn the representation associated with the normal condition, it will be capable to detect abnormal conditions. In practice this happens because an autoencoder is a neural network trained to try to copy its input x to its output y. Internally it has a hidden layer (or multiple layers) h that describes the representation of the data taken as input. An autoencoder is composed by two subparts: an encoding function $h = f(x)$ and a decoding function that reconstructs the input $y = g(h)$. Typically, autoencoders do not simply learn the identity function $g(f(x)) = x$ but are designed not to be able to copy perfectly, for instance the dimension of the hidden layer can be smaller than the dimension of the input. Hence, the output of an autoencoder is generally different from its input and the difference between input and output is called *reconstruction error*.~~ **The critical assumption is that an autoencoder can learn the representation of the normal state of a supercomputer node (looking at the measured features); afterwards, the model can be used to notice representation changes that underlie anomalous conditions. This happens because an autoencoder is a neural network trained to copy its input x to its output y. Internally it has hidden layers h encoding the representation of the data. An autoencoder is composed by two subparts: an encoding function $h = f(x)$ and a decoding function that reconstructs the input $y = g(h)$. Typically, autoencoders do not simply learn the identity function $g(f(x)) = x$ but are designed not to be able to copy perfectly, thus the output of an autoencoder is generally different from its input; this difference is called *reconstruction error*.**

~~We use the reconstruction error to identify anomalies. Although the error is not going to be equal to zero, an autoencoder can be trained to lower the discrepancy between input and output. In doing so, it learns the relationships among the features which form the input set. If new, previously unseen, data is given to the trained autoencoder as input, it will be capable to reproduce it accurately (with a low reconstruction error), as long as the new data is similar to the one found in the training set – the correlation between the features of the new data must be similar to the one learned by the autoencoder. If this condition does not hold, the autoencoder will not be able to correctly reconstruct the input and the reconstruction error will be greater. In [7] we demonstrate that we can detect anomalies by noticing when the reconstruction error is larger than the error computed with normal data. The experimental results clearly show that the reconstruction error is remarkably greater when the autoencoder is fed with previously unseen data that correspond to anomalous behaviour compared to unseen data corresponding to normal periods.~~ **The reconstruction error is used to identify anomalies. During training, an autoencoder learns the relationships among the features in the input set. If new, previously unseen, data is given to the trained autoencoder as input, it can reproduce it accurately (with a low reconstruction error), as long as the new data is similar to the one encountered in the training set – the correlation between the features of the new data must be similar to the one learned by the autoencoder. If this condition does not hold, the autoencoder cannot correctly reconstruct the input and the reconstruction error is greater.**

~~The overall scheme of our approach is depicted in~~ **The overall scheme of the approach is depicted in** Figure 1 (indicated by the numbered circles). The autoencoder-based model (yellow rectangles called **AE Model** and **Trained AE** in the figure) is trained using the data collected by the monitoring infrastructure installed in the target supercomputer (further details on the autoencoder are provided in Section 5). The training phase exploits normal data collected in a sufficiently long period of time (see Sec. 5.6 for practical guidelines); the training happens offline without real-time requirements. ~~This phase can take place in any machine; for instance, we used one of the supercomputer nodes to train the model. Once the model has been trained, it can be used to detect anomalies; if the model is lightweight it can be directly deployed on physical node that hosts the monitoring infrastructure (the first front-end containing also MQTT broker~~

~~and data bases in D.A.V.I.D.E.).~~ **The training phase can take place in any machine; for the proposed approach one of the supercomputer nodes was used. The trained models were then directly deployed on the node that hosts the monitoring infrastructure.** The model must be not too computationally demanding since it must be computed for each node and should not create an excessive overhead that would disrupt the data collection task. Having the autoencoder-based model on the same machine of the monitoring infrastructure reduces the storage requirements (once an anomaly has been detected unnecessary data can be discarded) and eliminates communication costs (since both data and detection model share the same host). **The goal of having a lightweight model guided the design of the autoencoder networks (topology, parameters, etc); the details will be provided in Sec. 5.4, where the implementation and experimental methodology are described.**

4.1. General Model VS Node-Specific Models

~~As previously stated, in our approach we use an autoencoder-based model to learn the normal behaviour of a supercomputer node. Since HPC systems are composed by clusters of nodes, an aspect that has to be specified is the number of different models that are needed. For example, is it better to have a model for each node in the cluster, where each model is a particular autoencoder neural network with its own hyperparameters, or rather to have a unique model to be applied to each node? To rephrase, we need to understand the trade-off between I) node-specific models and II) a unique general model.~~ **The proposed approach focuses on single HPC nodes, however supercomputers are composed by clusters of nodes – a strategy to tackle this issue must be defined. For instance, is it better to have a model for each node in the cluster, where each model is a particular neural network with its own hyperparameters, or rather to have a unique model to be applied to each node? To rephrase, it is crucial to understand the trade-off between I) node-specific models and II) a unique general model.**

In general, from the engineering perspective, it would be easier and more practical to have a single model that can be applied to all nodes, since this would simplify the training phase and deployment. However, there are some doubts that need to be taken into account. Has a general model the same detection accuracy of a set of models dedicated to each node? Are node-specific models prone to overfitting? How to train a general model? Sec. 5.4 will show the comparison between the different types of autoencoders and will answer these questions.

5. Experimental Evaluation

~~In this section we discuss the results of the autoencoder-based approach for anomaly detection. We first describe how we injected controlled anomalies in the supercomputer and then we show how our approach can detect them with high accuracy. We then compare the accuracy of our method with other techniques for semi-supervised fault detection taken from the literature (we are not aware of such techniques actually deployed on real HPC systems). Finally, we evaluate the impact of the training set size on the accuracy of the proposed approach.~~ **This section assesses the accuracy of the autoencoder-based approach for anomaly detection, by describing the controlled way to inject anomaly and the detection results. The accuracy of the proposed approach is compared with semi-supervised techniques from the literature. Finally, the impact of the training set size is evaluated.**

5.1. Anomaly Injection

Many kinds of sources can lead to anomalous behaviour and faulty states in HPC nodes. ~~In this work we focus on a specific type of anomaly that can often and easily arise in real supercomputers~~ **This work focuses on a specific type of anomaly that can often arise in real supercomputers**, namely *misconfiguration* at the single node level ~~.To be precise, we take into consideration the case of wrongly configured computing nodes.~~ – **more precisely, wrongly configured computing nodes.** Nowadays, most Linux systems (such as D.A.V.I.D.E.) can handle different operating modes that decide how the clock speed of the CPU is managed. This mechanism is allowed by a kernel-level driver typically referred to as

frequency governor [49]; different modes (or policies) can be specified and each one of them has a different impact on the clock speed, frequency and power consumption of the CPUs – given the same computational load, the power consumption of a supercomputer node can vary according to the frequency governor value.

~~For our experiments we used four different policies. The first one, *conservative*, is the default policy on D.A.V.I.D.E. and for our purpose it corresponds to the normal data used during the training phase.~~ **Four different policies were selected. The first one, *conservative*, is the default policy on D.A.V.I.D.E. and it corresponds to the normal data used during the training phase.** The conservative policy sets the CPU frequency according to the current workload. The anomalies were injected in the test set by changing the frequency governor to a non-default mode. ~~We opted for three policies~~ **The three anomalous policies are**: 1) *powersave*, that statically sets the CPU frequency to the lowest frequency available; 2) *performance*, that sets the CPU frequency to the highest value allowed in the frequency range; 3) *on-demand*, that scales the frequency according to the workload similarly to the conservative policy, but is more "aggressive", as it jumps to the highest frequency and then possibly backs off as the idle time increases. These are not hard failures, i.e. even with the wrong configuration the system remains operational. Hence, they are more subtle to track than hard failures. However, they do impact performance and they are representative of classes of misbehavior that decrease the efficiency of a machine: tracking down slower than expected nodes is a high-priority goal in large-scale systems, as slow nodes may severely impact service latency [50, 51, 52]. ~~Our~~**The** experiments were conducted on a subset of D.A.V.I.D.E. nodes (precisely 24 nodes, slightly more than half of the system), whose frequency governor policies were changed for periods of time ranging from dozens of minutes to a couple of days.

5.2. Experimental Setup

~~Since we were not allowed to introduce additional anomalies other than those described in Section5.1 and it was not easy to obtain anomaly data from the supercomputer in production, we decided to adopt an offline experimental setup, to better highlight the quality of the results and to operate in a controlled setting. In practice, we collected the measurements gathered by the monitoring infrastructure in several months of activity; these measurements (the features) form our main data set. Since neural networks work better with normalized values([53, 8]) we preprocessed the data to normalize them in the range $[0, 1]$, by removing the mean and scaling to unit variance. In an online setting the standardization process could be performed on the real-time new data by storing the mean and variance computed on the training set (the training of the model would still happen offline).~~ **The experiments were performed in an offline setup, not to hinder the supercomputer in production and to operate in a controlled setting. Several months of D.A.V.I.D.E. activity were collected and used as the main data set; the collected features were normalized in the range $[0, 1]$, by removing the mean and scaling to unit variance, as neural networks work better with normalized values[53, 8]. In an online setting the standardization process could be performed on the real-time new data by storing the mean and variance computed on the training set (the training of the model would still happen offline).**

The normalized data set was then divided in three parts: I) the training set DS^{Tr}, composed exclusively by examples belonging to periods of normal behaviour (with no anomalies injected); II) the test set without anomalies DS_N^{Te}~~, again containing examples from normal periods~~; III) the test set with anomalies DS_A^{Te}, with examples drawn solely from periods corresponding to anomaly injections. The training set DS^{Tr} is used to train the autoencoder ~~to learn the normal behaviour~~ while the two different test sets serve to assess the accuracy of our method. The data used to create these three sets correspond to a 83-days long period of D.A.V.I.D.E. lifetime (in production phase), comprising March, April and May 2018. During most of this time D.A.V.I.D.E. was in the "normal" state (frequency governor set to conservative), precisely for 66 days (80% of the time); in the remaining 17 days anomalies were injected by changing the frequency governor for periods ranging from hours to days. DS_A^{Te} corresponds to the 17 days with anomalies. ~~We randomly split the remaining days~~ **The remaining days were randomly split** between DS^{Tr} and DS_N^{Te}, with 80% of the data to the former and 20% to the latter.

5.3. Detection Accuracy

~~A point not addressed so far is how to use the reconstruction error to identify faulty states. In order to answer to this question there is one important observation to make: the reconstruction errors generated by examples from the test set without anomaly (DS_N^{Te}) follow a distribution very different from the errors obtained with the test set with anomalies (DS_A^{Te}).~~ **As mentioned in Section 4, faulty states are identified using the reconstruction error, thanks to the observation that the reconstruction errors obtained with the test set without anomaly (DS_N^{Te}) follow a distribution very different from those generated by the test set with anomalies (DS_A^{Te}).** The reconstruction error of an autoencoder with multivariate input and output[3] can be computed in different ways. ~~For example, let assume that the input and output vectors contain NF features. As said in Sec. 4, the reconstruction error is the difference between the output and the input; this difference is usually computed feature-wise and then averaged (also referred to as *normalized*). In our case, after a preliminary empirical evaluation when we experimented with different error types, we opted to use instead the *maximum* error.~~ **As said in Sec. 4, the reconstruction error is the difference between the output and the input; this difference is usually computed feature-wise and then averaged (also referred to as *normalized*). Alternatively, the *maximum* error can be used; in this paper, this was the chosen error, after a preliminary exploration.** The exact formula is the following:

$$E_i^{max} = \max_{j \in NF}(Y_j^i - X_j^i) \tag{1}$$

where X is the input vector and Y is the output vector; i is an index indicating the example in the data set and j is an index ranging among all features NF. Using this type of error instead of the normalized one leads to more diverse errors distributions for the different data sets.

~~In Figure 2 we can observe the errors distributions for some of the nodes that were injected with anomalies; we do not report all nodes since the figure would be unreadable and also because the excluded nodes have the same diverging distributions.~~ **Figure 2 displays the errors distributions for some of the nodes that were injected with anomalies; not all tested nodes were included to increase readability.** The errors distributions are presented as histograms: on the x-axis there is the value of the reconstruction error and on the y-axis there is the number of examples with corresponding error. The error reported is the one obtained by using node-specific autoencoders (see Sec. 4.1 and Sec. 5.4 for details). The nodes reported in the figure are *davide17*, *davide27*, *davide28*, *davide42* and *davide45*; these nodes were injected with two types of anomalies: frequency governors powersave and performance. The blue bars correspond to the distribution of the errors obtained with the normal test set while the red bars are the errors generated by the examples in the anomaly test set. The nodes can be distinguished by the different hues ~~(i.e. *davide28* has a dark tonality and *davide42* has lighter tones)~~. **To increase clarity, Figure 3 reports the error distributions for two nodes, *davide27* and *davide45*, with the distribution of normal errors in blue and anomaly distribution in red.**

~~At first glance we can see that the distribution of the errors obtained with the normal test set are very different from those obtained with the anomalous test set. For example, most of the normal errors are lower than 0.1 (actually the majority is even smaller than that) while the anomaly-generated errors are higher. This suggested us that this key difference could be used to devise an anomaly detection criteria.~~ **It is clear that normal test set and anomalous test set have very different errors distributions; most of the normal errors are lower than** 0.1 **(the majority is even smaller than that) while the anomaly-generated errors have higher values. This crucial observation suggests a criterion to detect anomalies.** ~~In practice, for each node we choose a threshold Θ;~~ **In practice, for each node a threshold Θ is selected;** whenever an unseen example is fed to the autoencoder-based model the reconstruction error is computed (according to Eq. 1): if the corresponding error is greater than Θ the example is classified as anomalous, normal otherwise. It remains to be decided how to choose the value of Θ; moreover, a preliminary analysis shown that different nodes can have different optimal values for this

[3]The input is composed by the vector with all the features collected by the monitoring infrastructure

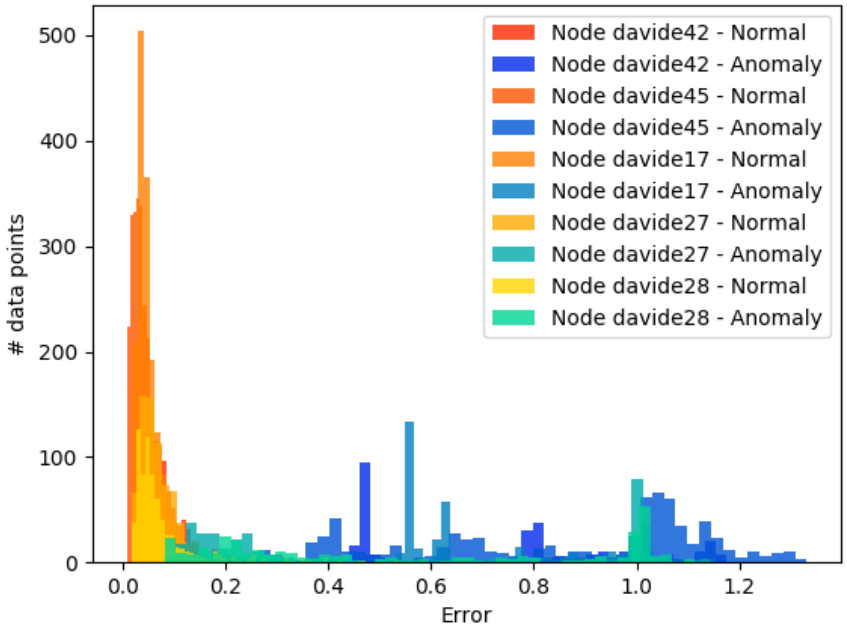
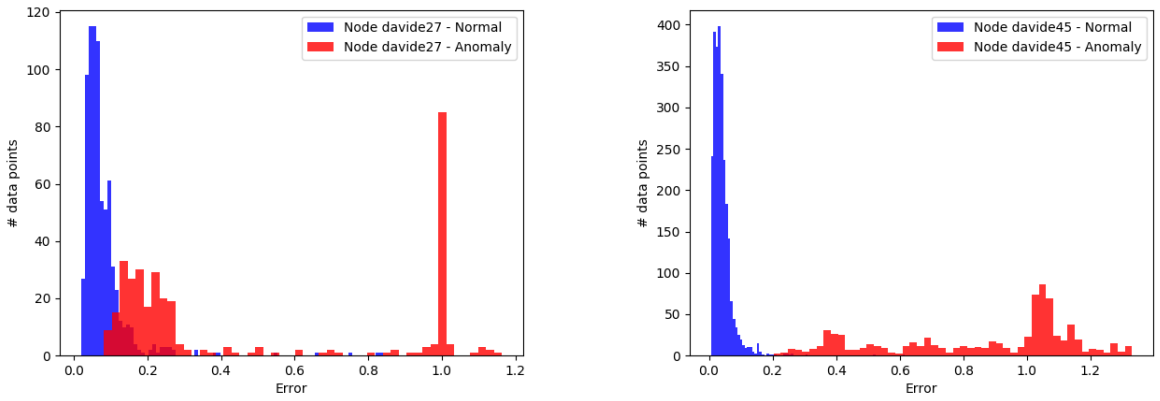


Figure 2: Autoencoder reconstruction error distributions of several nodes; normal examples (yellow/orange hues) and anomalous examples (blue hues)



(a) *davide27* (b) *davide45*

Figure 3: Autoencoder reconstruction error distributions of two nodes; normal examples (blue) and anomalous examples (red)

parameter. ~~To decide the threshold Θ we look at the n-th percentile of the errors distribution of the normal data~~ **The value Θ can be decided by looking at the n-th percentile of the errors distribution of the normal data** set[4].

~~The quality of the n value is determined by the overall accuracy obtained in the anomaly classification task over all the examples in the test sets (DS_N^{Te} and DS_A^{Te}). To measure the accuracy we use the *F-score*[53],~~

[4]The n-th percentile is a statistics indicating the value below which a given percentage of observations in a group of observations falls. For example, the 90-th percentile is the value below which 90% of the observations may be found.

~~a widespread metric that can assume values in the $[0, 1]$ range, with values closer to 1 indicating a higher accuracy[5]. We computed the F-score for the two classes in our case, anomaly and normal, plus the average weighted F-score (the weighted score has been used for breaking ties). For each node then there is a simple algorithm which tries all possible values of n in the range $[70, 100]$; this range was selected after a preliminary analysis.~~ **The quality of the n value is determined by the overall accuracy obtained in the anomaly classification task over all the examples in the test sets (DS_N^{Te} and DS_A^{Te}); as accuracy measure, the *F-score*[53] was chosen (values in the $[0, 1]$ range, with values closer to 1 indicating higher accuracy). For each node, the optimal Θ has been found by varying n in the range $[70, 100]$ (values selected after a preliminary analysis) and by measuring the corresponding accuracy; then, n and related Θ with the highest F-score were picked.**

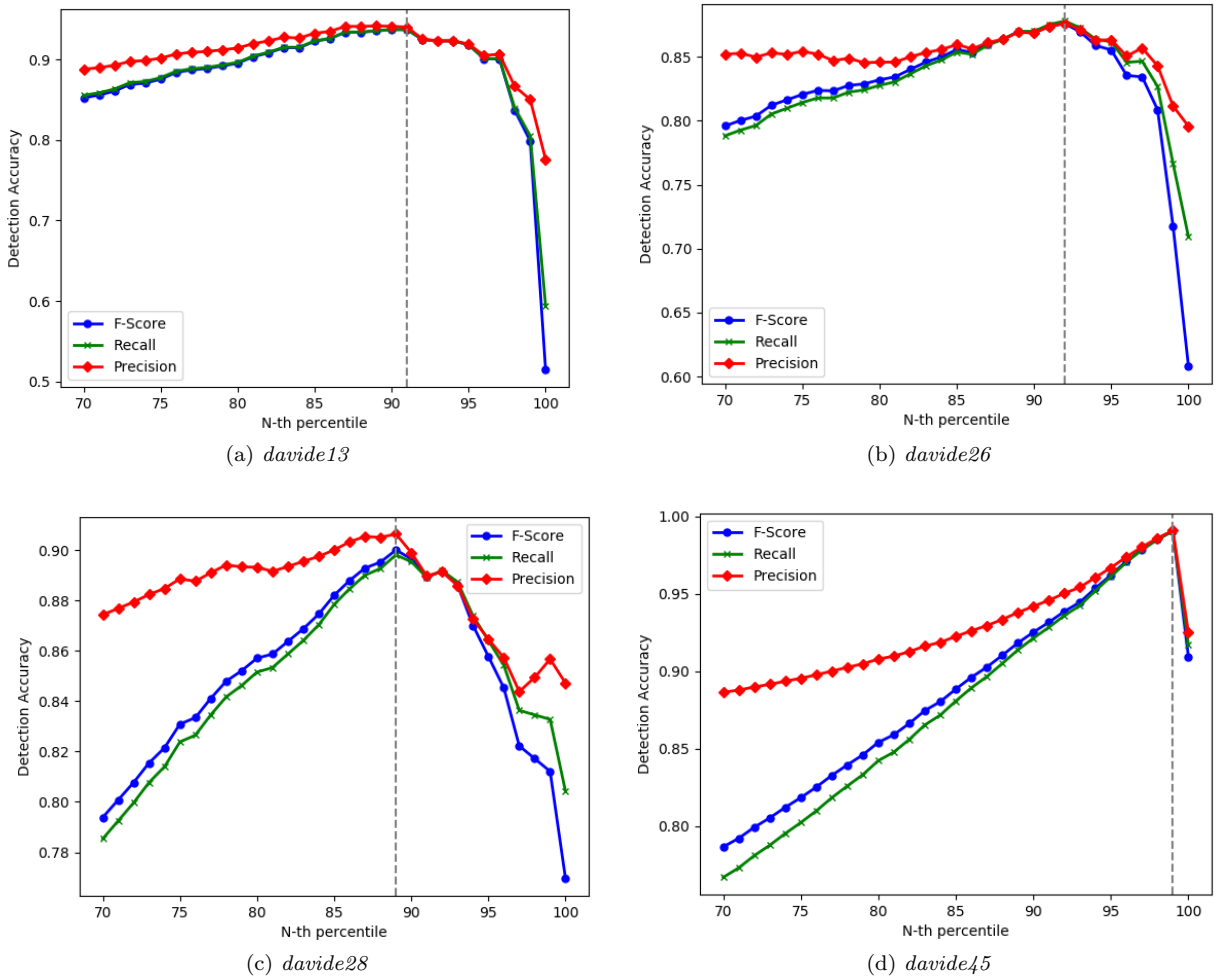
In the case of the generic model for all computing nodes the mechanism is totally analogous, with n and *Theta* chosen depending on the errors computed for DS_N^{Te} and DS_A^{Te}; the only difference is that for the general model the training set and test set are not specific for a node (details in Sec. 5.4). The best n value for the general model is equal to 94; for the node-specific models the best n can differ for each node, for example it is equal to 99 for nodes such as *davide16*, *davide29* and *davide45*, down to 80 for *davide12*, passing through intermediate values such as 89 for *davide28* and 91 for *davide13*. ~~In Figure 3 we can observe how the variation of n (and hence *Theta*) affects the detection accuracy~~ n **and hence** *Theta* **are then obtained without the assistance of domain experts, but rather by looking at the errors distribution of the training set. Their optimal values are task (and node) specific and are computed via a simple exploration of the parameters space. Figure 4 shows the impact of these parameters on the detection accuracy;** on the x-axis there is n (in the range $[70, 100]$) while the y-axis reports the accuracy, measured as weighted F-score, precision and recall (drawn with three different lines and markers, respectively, blue dots, red diamonds and green crosses). The precision is inversely proportional to the number of false positives and the recall is inversely proportional to the number of false negatives; values closer to one indicate fewer false positives or negatives. The grey vertical dashed line pinpoints the best n for the corresponding node. The results of the node-specific autoencoders for 4 different nodes are shown, *davide13*, *davide26*, *davide28*, and *davide45*.

Selecting higher values of n means rising the threshold *Theta* used to discern anomalous points from normal ones. Higher values lead to a more conservative anomaly detection criterion – thus ~~we are~~ minimizing the number of false positives, at the expense of missing some anomalous points. This decreases the number of false alarms. However, parameters n and *Theta* allow more flexible and goal-oriented approaches: decreasing their values leads to a more sensitive anomaly detection, with fewer missed anomalies but with an increased number of false alarms. The best trade-off can be decided by system administrators and facility owners.

5.4. Comparison between different autoencoders

~~In this section we are going to compare two possible implementations of the autoencoder-based approach~~ **This section compares two types of autoencoder-based approach**: I) a set of node-specific autoencoders, on for each node in the HPC cluster or II) a single general model. ~~In the first case, we create and train an autoencoder-based model for each node of the system; the training and test set are drawn from the corresponding data set of that node. In the general case, we create a single autoencoder trained with normal data coming from multiple nodes, namely the normal data collected on all the nodes involved in our experiments; clearly this leads to a much larger training set compared to those used for the node-specific models and the training time increases as well. After the training phase, we test the general model by feeding it with the test set of a single node (whose normal data was part of the training set as well); the tests were repeated on all nodes.~~ **In the first case, each node has its own autoencoder; training and test set are drawn from the data set of the node. In the general case, a single autoencoder is trained with normal data coming from multiple nodes; clearly, this leads to a much larger training set compared to the node-specific ones and the training time increases as well. After the training, the general model is evaluated by feeding it with the test set of a single node (whose normal data was part of the training set as well); the tests were repeated on all nodes.**

All autoencoders share the same network configuration. ~~We decided to employ a~~ **They have a** fairly simple structure composed by three ~~main~~ layers: 1) an input layer with as many neurons as the number



(a) *davide13* (b) *davide26*

(c) *davide28* (d) *davide45*

Figure 4: Node Specific Models – Detection Accuracy VS n-th percentile for nodes *davide13*, *davide26*, *davide28*, and *davide45*

of features; 2) a densely connected intermediate sparse layer[54, 55] with a number of neurons equal to the number of feature multiplied by ten, with Rectified Linear Units[56] (*ReLu*) as activation functions and a L1 norm regularizer[57, 8]; III) a final dense output layer with the same number of neurons as the input layer and with linear activations. ~~To reduce overfitting we applied a *Dropout* layer~~ **To reduce overfitting, a *Dropout* layer was applied** [58] ~~(dropout rate equal to 0.2)~~ to the output of layers 1 and 2 **(dropout rate equal to** 0.2**)**. Each autoencoder is trained with *Adam* optimizer[59] 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). **The hyperparameters for the Adam optimizer were: learning rate** 0.001**,** $\beta_1 = 0.9$**,** $\beta_2 = 0.999$**, decay** 0.0**.** 10% of the training set was used for validation. This network topology and hyperparameters values were obtained after a preliminary empirical evaluation, following the criteria of finding a good trade-off between accuracy and complexity of the model. Simpler models have lower computational and memory demands and can be used for real-time inference without degrading the supercomputer performance or incurring in an additional cost in the computing nodes' telemetry system. For the training, one of the nodes of D.A.V.I.D.E. itself was used; the average training time was 65.5 seconds for the node-specific models, while the time required to train the general model was 672 seconds (slightly more than 11 minutes). **For test phase, the trained models were loaded on the embedded monitoring boards to perform detection (offline fashion, for the moment). Due to the relatively simple topology of the**

Node	*Anomaly Type*	*Dedicated Models*			*General Model*		
		F-Score N	F-Score A	F-Score W	F-Score N	F-Score A	F-Score W
davide10	ondemand	0.965	0.883	0.946	0.923	0.621	0.853
davide11	ondemand	0.973	0.937	0.963	0.999	0.487	0.869
davide12	ondemand	0.704	0.774	0.747	0.893	0.835	0.874
davide13	ondemand	0.956	0.962	0.959	0.97	0.881	0.934
davide16	powersave	0.994	0.97	0.99	0.999	0.82	0.97
davide17	powersave	0.994	0.977	0.991	0.999	0.879	0.987
davide18	powersave	0.994	0.973	0.99	0.984	0.838	0.957
davide19	powersave	0.978	0.999	0.99	0.999	0.926	0.999
davide26	performance	0.904	0.708	0.846	0.949	0.558	0.828
davide27	performance	0.925	0.842	0.9	0.986	0.782	0.923
davide28	performance	0.92	0.822	0.892	0.996	0.72	0.914
davide29	performance	0.99	0.863	0.981	0.995	0.574	0.963
davide42	powersave	0.993	0.983	0.99	0.99	0.853	0.952
davide45	powersave	0.956	0.962	0.99	0.979	0.823	0.943
Average	NA	0.943	0.898	0.935	0.956	0.726	0.919

Table 1: Comparison between node-dedicated and general models

autoencoder networks, even with the limited computing capacity of the monitoring boards, the inference time (the time required to process an example from the test set) was very short, around 0.011 **seconds – for both node-specific and general models. This is a negligible time w.r.t. the sampling rate of 5 (or 10) seconds.**

~~In Table 1 we see the results of the experiments~~ **Table 1 reports the results of the experiments** comparing node-specific and general models. The first column from the left indicates the node and the second column (*anomaly type*) specifies the kind of anomaly injected in the node. We selected a subset of all nodes that were injected with anomalies in order create a more compact view – the remaining tested nodes (*davide4*, *davide5*, *davide29*, *davide30*, *davide31*, *davide32*, *davide33*, *davide34*, *davide35*, *davide36*) presented similar numbers; the final row, *Average*, represents the average values computed on all nodes used for the experiments. The rest of the table reports the F-scores for all classes (F-scores N, A, and W correspond, respectively, to the F-score computed on the normal test set, the anomaly test set and the weighted F-score) and for both the dedicated models (from the third column to the fifth one) and the general model (the last three columns). The values reported are the average obtained after repeating the training and test phases for 10 times.

The first thing to be noticed, is that, on average, the accuracy is pretty good, as highlighted by the weighted F-scores higher than 0.91 (that corresponds to an accuracy higher than 90%). In general, the F-scores for the anomaly class are significantly smaller than the normal class; this is due to the relatively higher number of false negatives ~~(anomalous examples classified as normal)~~ compared to the number of false positives ~~(normal examples classified as anomalies)~~. If the system administrators decide that is more important not to miss any anomaly and hence to increases the accuracy detection of anomalous examples ~~(at the expense of a lower overall F-score and more false positives)~~, a simple but effective strategy is to lower the threshold used for the classification ($Theta$, modulated by varying n).

As a second observation, ~~we can notice how~~ different accuracies are generally obtained with nodes with different types of injected anomalies. Nodes whose frequency governor policy was changed to powersave have an average weighted F-score equal to 0.99 with dedicated models and 0.968 with the general model; the average values for nodes whose policy was changed to performance are 0.905 and 0.907, and finally for those nodes with ondemand anomalies the average values are 0.904 and 0.883. It appears that all models are better at detecting anomalies of the powersave type; this can be explained by thinking at the peculiar characteristics of a supercomputing cluster. The vast majority of the software running in HPC systems are scientific applications and numerical computations; these are generally very CPU-intensive applications

that run for most of their duration at the highest possible frequency allowed by the clock of computing units. Hence, the performance frequency governor policy resembles the normal operational behaviour of a supercomputer node, at least compared to the powersave mode which decreases the frequency to the minimum allowed one. Similarly, setting the frequency governor to ondemand policy leads to a state that is not too different from the normal one; clearly, non-negligible differences still remain and that is the reason why anomalies are still detected.

Finally, ~~we can observe that~~ there is a very moderate difference between the dedicated models and the general one. As one could have expected, the node-specific models have a slightly higher accuracy, with an average weighted F-score equal to 0.935 compared to 0.919, that is a 1.7% difference. In practical terms, this result suggests that the general model can be adopted without losing too much accuracy, if the easier deployment is to be preferred. It can be worth pointing out that while the time needed to train the single general model is longer than the average training time of node dedicated models (more or less eleven times longer), ~~if we add all the times for each node-specific autoencoder we end up with a larger overall value w.r.t. the training time of the single model.~~ **training all node-specific models takes longer than training the general one.**

5.5. Comparison with the state-of-the-art

As discussed in Section 2, the vast majority of techniques for anomaly detection in HPC systems and data centers belongs to the supervised methodology. For instance, the current state-of-the-art (to the best of our knowledge) is the approach proposed by Tuncer et al.[11]~~, that relies on a classifier composed by a features extraction component plus a Random Forest model~~. ~~Our~~**The** approach falls in an entirely different category ~~(the semi-supervised methodology)~~, hence a direct comparison with this technique is not fair~~, as it assumes the availability of a training dataset that is much more difficult to collect than ours, since it has to include many instances of labeled anomalies in production~~. ~~Nevertheless, we applied the Tuncer technique to our problem in order to show that our method has an accuracy that is very close to the current state-of-the-art, without having all the limitations related to the supervised training.~~ **Nevertheless, applying Tuncer's technique to the problem considered in in this paper shows that the proposed method has an accuracy very close to the current state-of-the-art, without having all the limitations related to the supervised training.** The average accuracy over all nodes with the supervised method is very high, with an average weighted F-score equal to 0.99.

~~To have a more comprehensive comparison, we also selected a subset of~~ **For a more comprehensive comparison, a subset of** techniques for semi-supervised anomaly detection found in the literature **has been selected**~~; these algorithms have not been applied to supercomputers yet, thus we performed our analysis by applying them to our problem~~. The experimental setup is the same used for ~~our~~**the** autoencoder-based model: during the training phase the algorithms from the literature are taught to recognize the normal behaviour of a HPC cluster node (using the training set DS^{Tr}); afterwards, they were fed with previously unseen input, both normal and anomalous (the test sets DS_N^{Te} and DS_A^{Te}), and ~~we compute~~ the detection accuracy **was computed** ~~using the F-score (normal class, anomaly class and weighted). For these experiments we opted for the node-dedicated model, and for every algorithm we trained a distinct model for each node.~~ **For every algorithm a dedicated approach was chosen (a distinct model for each node.**

In general, semi-supervised methods for anomaly detection ~~employ an idea similar to ours:~~ identify anomalies by exploiting their dissimilarity from the normal state. They are based on learning the normal behaviour of the target system, via ML or statistical models; faults are then detected because they present a different signature w.r.t. the learnt one. ~~These methods are typically used for outlier or novelty detection~~. The most common fall in one ~~(or multiple)~~ of the following categories: probability density estimation [60, 61, 62, 63], one-class Support Vector Machine (SVM)[64, 65], elliptical envelope [66, 67], Isolation Forest[68, 69] and neighbourhood identification [70, 71]. Since there is no clear technique outperforming all the others[72, 73] ~~(to the best of our knowledge)~~, ~~we implemented~~ a subset of algorithms from the literature **were implemented**~~(a few from all the different areas aforementioned)~~. All the alternative algorithms were implemented in Python, using the ML and statistics module *scikit-learn*[74]; all non-specified parameters were set to their default values, as defined in the library.

Node	*GMM*				*EE*	*IF*	*SVM*		*AE*	
	Diag	Spher	Tied	Full			Poly	RBF	Dedicated	General
davide10	0.916	0.927	0.924	0.922	0.911	0.941	0.262	0.811	**0.946**	0.853
davide11	0.91	0.909	0.921	0.923	0.801	0.946	0.237	0.65	**0.963**	0.869
davide12	0.864	0.865	**0.887**	0.191	0.154	0.278	0.216	0.608	0.747	0.874
davide13	0.897	0.876	0.892	0.904	0.434	0.953	0.16	0.66	**0.959**	0.934
davide16	0.854	0.515	0.855	0.884	0.915	0.923	0.606	0.926	**0.99**	0.97
davide17	0.272	0.267	0.269	0.509	0.914	0.929	0.613	0.931	**0.991**	0.987
davide18	0.882	0.858	0.888	0.875	0.715	0.923	0.614	0.933	**0.99**	0.957
davide19	0.887	0.523	0.524	0.909	0.762	0.919	0.624	0.941	0.99	**0.999**
davide26	0.893	**0.895**	0.894	**0.895**	0.376	0.701	0.218	0.609	0.846	0.828
davide27	0.165	0.162	0.161	0.922	0.825	0.652	0.389	0.656	0.9	**0.923**
davide28	0.926	0.756	0.788	**0.939**	0.773	0.912	0.406	0.635	0.892	0.914
davide29	0.843	0.841	0.799	0.842	0.882	0.89	0.455	0.92	**0.981**	0.963
davide42	0.722	0.356	0.718	0.295	0.793	0.853	0.727	0.935	**0.99**	0.952
davide45	0.394	0.561	0.518	0.752	0.627	0.67	0.661	0.933	**0.99**	0.943
Average	0.745	0.665	0.717	0.769	0.706	0.821	0.442	0.796	**0.935**	0.919
Opt. gap (%)	24.8	32.8	27.6	22.3	28.7	17.1	55.3	19.6	**5.6**	7.2

Table 2: Comparison with state-of-the-art

~~In Table 2 we report~~**Table 2 reports** the weighted F-scores for the algorithms from the literature (columns from 2 to 9) and the weighted F-scores obtained with the proposed autoencoder-based method (column 10, node-specific approach, and 11, general model). Each row of the table corresponds to a node (again a subset of all nodes where anomalies were injected). ~~We do not report the F-scores for the Tuncer et al.~~ **F-scores for the Tuncer's method are not reported**~~method [11]~~ because its accuracy is extremely high for all nodes, ranging from 0.990 to 0.999~~; hence the table shows only the results for semi-supervised techniques, both ours and from the literature~~. From left to right the algorithms are the following. Gaussian Mixture Models (*GMM* in the table) are probabilistic models that assume that all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters. During the training phase the parameters can be estimated from a training data set using the Expectation-Maximization (EM) algorithm, hence modeling the structure of the normal state of the supercomputer nodes. Once trained, ~~we can give~~ new data **can be fed** as input to the GMM and it will generate a corresponding probability; a high value indicates that the new example has the same distribution seen during the training (normal state), otherwise it is probably an anomaly. ~~We implemented GMM~~**GMM was implemented** with four different types of covariance matrices: diagonal (*Diag* in the table), spherical *Spher*), tied and full.

The Elliptical Envelope (*EE*) algorithm models the data as a high dimensional Gaussian distribution with possible covariances between feature dimensions. It attempts to find an boundary ellipse that contains most of the data. Any data outside of the ellipse is classified as anomalous. The Isolation Forest (*IF*) algorithm is based on random forests; it is particularly effective for high-dimensional data sets. The ensemble (a forest of decision trees) "isolates" observations by randomly selecting a feature and then a split value between the maximum and minimum values of the feature. The number of split required to isolate a sample is equivalent to the path length from the root node to the terminating node. This length (average over a forest of trees) measures the normality of an example. Random partitioning produces shorter paths for anomalies. Hence examples that, once fed to the forest, generate shorter path length are highly likely to be anomalies.

A one-class Support Vector Machine (*SVM*) can be used to detect outliers in semi-supervised fashion. A SVM model is trained to learn a closed frontier around the training examples, which correspond to the normal state ~~in our case~~. Then new examples are classified as normal if they lay within the frontier-delimited subspace, anomalous (different from the original distribution) otherwise. ~~We use two~~**Two** different kinds of kernels **were used**, polynomial of third degree (*Poly*) and Radial Basis Function (*RBF*).

The last two rows correspond, respectively, to the average value computed over all nodes and the op-

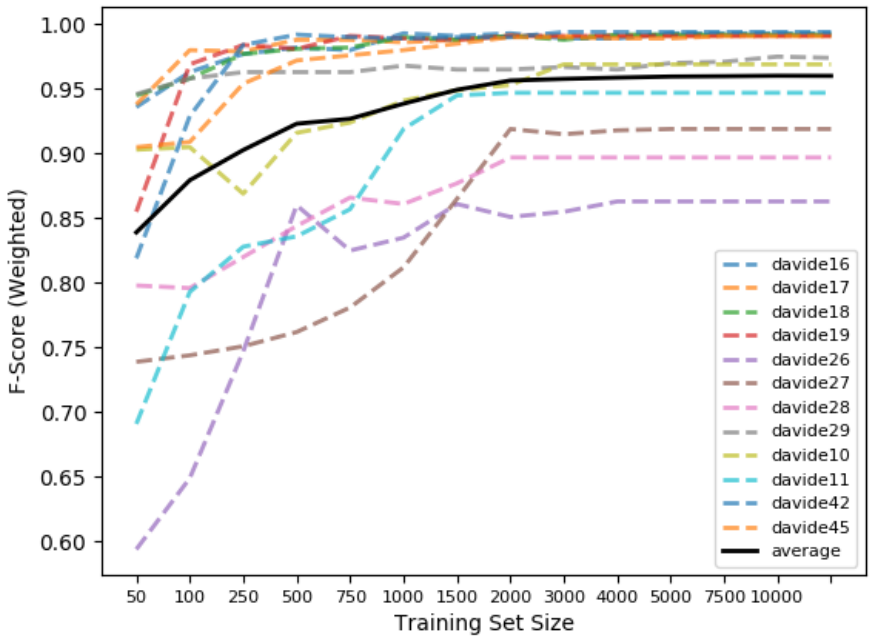


Figure 5: Training set size impact on the anomaly detection accuracy

timality gap, that is the distance (expressed as percentage) from the weighted F-score obtained with the supervised approach~~, that we consider as the optimal solution to our task~~ **(optimal solution)**; the optimality gap is computed only with the average weighted F-scores. It is very easy to see that the autoencoder-based models (both the node-specific ones and the general model) significantly outperform the methods from the literature, since all algorithms have accuracies lower than 90%, and the highest weighted F-score is reached by Isolation Forest with 0.821 – this is a value 10.6% lower than the general model and 12.2% lower than the average result for node-specific models. Furthermore, the optimality gap is much smaller for ~~our~~**the proposed** methods which, on average, have a F-score just 6.2% smaller than the supervised method, while the average distance for the literature methods is 28.5% and the best gap is obtained with Isolation Forest and is equal to 17.1% – more than twice the optimality gap with the autoencoder-based general model.

5.6. Data Set Size Impact

Finally, ~~in~~ this section ~~we analyse~~**analyses** the effect of the training set size on the anomaly detection accuracy. This is a very important aspect since it could be costly to gather enough data describing the normal state of a supercomputer. To be precise, the issue is not obtaining large amount of data ~~(given the availability of a monitoring infrastructure)~~ but rather to be sure that in a given period no faults or anomalous conditions happened. ~~In order to evaluate this issue we performed experiments with varying sizes of training set; in practice we reduced the size of the original training set~~ **This analysis was performed by varying the size of training set; in practice the size of the original training set was reduced** ~~(the one obtained from the 66-days period described in Sec. 5.2)~~ by drawing random subsets with smaller sizes.

Figure 5 shows the impact of the size of the training set on the accuracy of the detection **(considering only node-specific models)**; on the x-axis there is the size of the training set while the y-axis reports the weighted F-score obtained with the model trained with the corresponding training set size – the size is measured as the number of examples in the data set (i.e. the number of data points collected by the monitoring infrastructure in the selected period). ~~We report only the results obtained with the node-Specific models.~~ Each dashed line represents a node ~~(again, in order not to clutter the figure we plot only a subset of nodes)~~; the black solid line is the average computed over all nodes that were injected with anomalies. The original training set size corresponds to roughly 12k examples (or around 8 weeks of monitoring).

As seen in the previous section, the results differ significantly ~~if we look at~~**for** different nodes; ~~we see the confirmation that~~ nodes with anomalies of powersave type (such as *davide17* and *davide45*) have higher accuracies than nodes with performance or ondemand frequency governors (i.e. node *davide26*, light violet line, or *davide11*, light blue line). ~~We also notice that for~~**For** certain nodes, a larger number of examples is needed to reach good level of accuracies – let assume that 90% is a good level of accuracy w.r.t. the state-of-the-art (see Sec. 5.5). However, ~~if we consider the average value over all nodes (the black solid line), we see that F-score higher than 0.9 are reached with relatively small data set size, around~~ **looking at the average value over all nodes (black solid line), F-scores higher than 0.9 are reached with relatively small data set size, around** 500 examples, which correspond to about 42 hours of continuous monitoring and gathering of data. ~~If we collect more data (that is, data that the system administrators can guarantee that it belonged to a system operating in normal state)~~ **With more normal training data** higher accuracies can be obtained, for example a data set size of 1k samples (around 3 and a half days) corresponds to an average F-score equal to 0.938, a very accurate detection rate.

6. Conclusion

This paper presented an approach to solve the problem of automated anomaly detection in High Performance Computing systems. This is a very complex task with high relevance in real-world supercomputers, since nowadays the vast majority of anomaly and fault diagnosis is performed by system administrators. The **proposed** approach ~~we propose~~ belongs to the semi-supervised Machine Learning field, in contrast with most of the state-of-the-art and techniques from the literature that rely on supervised scheme. The latter methodology is very common but not easily employed in the supercomputing setting, where it is not easy to find or to build labeled and unbiased data sets to be used for training.

~~With our approach we overcome this limitation since we focus on learning only~~ **The proposed approach overcomes this limitation as it focuses on learning only** the behaviour of a HPC cluster in its normal state, thus requiring only to identify and monitor a limited period of time when the supercomputer was operating without faults. ~~For this reason, we are able to develop a high-accuracy detection method and to apply it directly to a HPC machine in production. We also considered two different kinds of models~~ **For this reason, a high-accuracy detection method was developed and applied to a HPC machine in production. Two two different kinds of models were considered**, one where each cluster node has its own specific autoencoder-based model and one where a single model is applied to all nodes. The former models have higher accuracy while the general one is more easily deployable. ~~We also studied the relation between size of training set and accuracy of the results; we found that, in order to have good accuracies on most of the HPC cluster nodes, four days of monitored data are sufficient.~~ **In order to have good accuracy on most of the HPC cluster nodes, four days of monitored data are sufficient.**

~~We compare our method with other algorithms coming from the semi-supervised area~~ **The proposed method was compared with other algorithms coming from the semi-supervised area** (never tested before on the HPC settings) and **the results** show that ~~our~~**the autoencoder-based** approach significantly outperforms them, with an improvement in accuracy as high as 12%. ~~We also compare the results of all semi-supervised techniques (ours included) with the best current algorithm for HPC anomaly detection, belonging to the supervised class.~~ **A comparison among all semi-supervised techniques (including the approach described in this paper) and the best current algorithm for HPC anomaly detection was carried out.** The comparison is based on an unfavorable setting for semi-supervised methods, but nevertheless ~~we~~experimental results show that ~~our approach~~**the autoencoder model** has an accuracy very similar to the state-of-the-art supervised techniques, with a decrease of only 5.6%, while for the rest of semi-supervised methods there is a performance decrease of at least 17% (and usually much worse).

Acknowledgements

We want to thank CINECA and E4 for granting us the access to their systems.

[1] J. L. Hennessy, D. A. Patterson, Computer Architecture, Sixth Edition: A Quantitative Approach, 6th Edition, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2017.

[2] X. Yang, Z. Wang, J. Xue, Y. Zhou, The reliability wall for exascale supercomputing, IEEE Transactions on Computers 61 (6) (2012) 767–779.
[3] T. M. Mitchell, Machine learning and data mining, Communications of the ACM 42 (11) (1999) 30–36.
[4] T. Tommasi, N. Patricia, B. Caputo, T. Tuytelaars, A deeper look at dataset bias, in: Domain Adaptation in Computer Vision Applications, Springer, 2017, pp. 37–55.
[5] A. Netti, Z. Kiziltan, O. Babaoglu, A. Sirbu, A. Bartolini, A. Borghesi, Finj: A fault injection tool for hpc systems, in: M. G. et al. (Ed.), Proceedings of Euro-Par 2018: Parallel Processing Workshops. Euro-Par 2018, Vol. 11339 of Lecture Notes in Computer Science, Springer, 2019, p. in press.
[6] R. S. Michalski, J. G. Carbonell, T. M. Mitchell, Machine learning: An artificial intelligence approach, Springer Science & Business Media, 2013.
[7] A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, L. Benini, Anomaly detection using autoencoders in high performance computing systems, in: AAAI, in press.
[8] I. Goodfellow, Y. Bengio, A. Courville, Y. Bengio, Deep learning, Vol. 1, MIT press Cambridge, 2016.
[9] Cineca inter-university consortium web site, http://www.cineca.it//en, accessed: 2018-06-29.
[10] O. Tuncer, E. Ates, Y. Zhang, A. Turk, J. Brandt, V. J. Leung, M. Egele, A. K. Coskun, Diagnosing performance variations in hpc applications using machine learning, in: International Supercomputing Conference, Springer, 2017, pp. 355–373.
[11] O. Tuncer, E. Ates, Y. Zhang, A. Turk, J. Brandt, V. J. Leung, M. Egele, A. Coskun, Online diagnosis of performance variation in hpc systems using machine learning, IEEE Transactions on Parallel and Distributed Systems-doi:10.1109/TPDS.2018.2870403.
[12] L. Breiman, Random forests, Mach. Learn. 45 (1) (2001) 5–32. doi:10.1023/A:1010933404324.
[13] P. Bodik, M. Goldszmidt, A. Fox, D. B. Woodard, H. Andersen, Fingerprinting the datacenter: automated classification of performance crises, in: Proceedings of the 5th European conference on Computer systems, ACM, 2010, pp. 111–124.
[14] Z. Lan, Z. Zheng, Y. Li, Toward automated anomaly identification in large-scale systems, IEEE Transactions on Parallel and Distributed Systems 21 (2) (2010) 174–187.
[15] E. Baseman, S. Blanchard, N. DeBardeleben, A. Bonnie, A. Morrow, Interpretable anomaly detection for monitoring of high performance computing systems, in: Outlier Definition, Detection, and Description on Demand Workshop at ACM SIGKDD. San Francisco (Aug 2016), 2016.
[16] M. C. Dani, H. Doreau, S. Alt, K-means application for anomaly detection and log classification in hpc, in: International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, Springer, 2017, pp. 201–210.
[17] M. Gabel, A. Schuster, R.-G. Bachrach, N. Bjørner, Latent fault detection in large scale services, in: Dependable Systems and Networks (DSN), 2012 42nd Annual IEEE/IFIP International Conference on, IEEE, 2012, pp. 1–12.
[18] B. Cai, Y. Liu, Q. Fan, Y. Zhang, Z. Liu, S. Yu, R. Ji, Multi-source information fusion based fault diagnosis of ground-source heat pump using bayesian network, Applied energy 114 (2014) 1–9.
[19] B. Cai, Y. Zhao, H. Liu, M. Xie, A data-driven fault diagnosis methodology in three-phase inverters for pmsm drive systems, IEEE Transactions on Power Electronics 32 (7) (2016) 5590–5600.
[20] B. Cai, Y. Liu, M. Xie, A dynamic-bayesian-network-based fault diagnosis methodology considering transient and intermittent faults, IEEE Transactions on Automation Science and Engineering 14 (1) (2016) 276–285.
[21] B. Cai, H. Liu, M. Xie, A real-time fault diagnosis methodology of complex systems using object-oriented bayesian networks, Mechanical Systems and Signal Processing 80 (2016) 31–44.
[22] Y. Wang, M. Liu, Z. Bao, S. Zhang, Stacked sparse autoencoder with pca and svm for data-based line trip fault diagnosis in power systems, Neural Computing and Applications (2018) 1–13.
[23] C. Shen, Y. Qi, J. Wang, G. Cai, Z. Zhu, An automatic and robust features learning method for rotating machinery fault diagnosis based on contractive autoencoder, Engineering Applications of Artificial Intelligence 76 (2018) 170–184.
[24] J. E. Siegel, S. Pratt, Y. Sun, S. E. Sarma, Real-time deep neural networks for internet-enabled arc-fault detection, Engineering Applications of Artificial Intelligence 74 (2018) 35–42.
[25] B. S. J. Costa, P. P. Angelov, L. A. Guedes, Fully unsupervised fault detection and identification based on recursive density estimation and self-evolving cloud-based classifier, Neurocomputing 150 (2015) 289–303.
[26] T. Ince, S. Kiranyaz, L. Eren, M. Askar, M. Gabbouj, Real-time motor fault detection by 1-d convolutional neural networks, IEEE Transactions on Industrial Electronics 63 (11) (2016) 7067–7075.
[27] D. Kwon, H. Kim, J. Kim, S. C. Suh, I. Kim, K. J. Kim, A survey of deep learning-based network anomaly detection, Cluster Computing (2017) 1–13.
[28] B. R. Kiran, D. M. Thomas, R. Parakkal, An overview of deep learning based methods for unsupervised and semi-supervised anomaly detection in videos, Journal of Imaging 4 (2) (2018) 36.
[29] M. Goldstein, S. Uchida, A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data, PloS one 11 (4) (2016) e0152173.
[30] W. A. Ahmad, A. Bartolini, F. Beneventi, L. Benini, A. Borghesi, M. Cicala, P. Forestieri, C. Gianfreda, D. Gregori, A. Libri, F. Spiga, S. Tinti, 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), 2017, pp. 964–973. doi:10.1109/IPDPSW.2017.22.
[31] E4 computer engineering, https://www.e4company.com/en/.
[32] Prace. partnership for advanced computing in europe.
[33] J. J. Dongarra, H. W. Meuer, E. Strohmaier, 29th top500 Supercomputer Sites, Tech. rep., Top500.org (Nov. 1994).
[34] W.-c. Feng, K. Cameron, The green500 list: Encouraging sustainable supercomputing, Computer 40 (12) (2007) pp. 50–55.
[35] F. Beneventi, A. Bartolini, C. Cavazzoni, L. Benini, 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, pp. 1038–1043.
[36] A. Bartolini, A. Borghesi, A. Libri, 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, Ischia, Italy, May 08-10, 2018, 2018, pp. 303–308. doi:10.1145/3203217.3205863.
[37] O. Standard, Mqtt version 3.1. 1, URL http://docs. oasis-open. org/mqtt/mqtt/v3 1.
[38] G. Coley, Beaglebone black system reference manual, accessed: 2019-01-03 (2013).
URL https://cdn-shop.adafruit.com/datasheets/BBB_SRM.pdf
[39] A. Libri, A. Bartolini, L. Benini, Dwarf in a giant: Enabling scalable, high-resolution hpc energy monitoring for real-time profiling and analytics, in: Supercomputing2019, in press.
[40] I. Corporation, User manual for automated measurement of systems for temperature and energy reporting, accessed: 2019-01-03 (2016).
URL https://github.com/open-power/amester/blob/master/vfs/doc/manual.txt
[41] M. Broyles, et al., Ibm energyscale for power8 processor-based systems, white paper (November 2015).
URL https://www-01.ibm.com/common/ssi/cgi-bin/ssialias?htmlfid=POW03125USEN
[42] T. Rosedah, C. Lefurgy, M. Broyles, Measuring and managing energy in openpower, in: International Conference on High Performance Computing, Springer, 2016, pp. 255–267.
[43] Ipmiutil user guide, accessed: 2019-01-03.
URL http://ipmiutil.sourceforge.net/
[44] I. Corporation, Intel ipmi technical resources, accessed: 2019-01-03.
URL https://www.intel.com/content/www/us/en/servers/ipmi/ipmi-technical-resources.html
[45] Kairosdb a fast scalable time series database, https://github.com/kairosdb/kairosdb, accessed: 2018-07-02.
[46] Apache, Apache cassandra, https://http://cassandra.apache.org/, accessed: 2019-01-04.
[47] A. Bartolini, A. Borghesi, A. Libri, F. Beneventi, Examon, accessed: 2019-01-03.
URL https://github.com/EEESlab/examon
[48] I. Corporation, Occ ipmitool sensors, accessed: 2019-01-03 (2016).
URL https://github.com/open-power/docs/blob/master/occ/OCC_ipmitool_sensors.pdf
[49] D. Brodowski, N. Golde, Cpu frequency and voltage scaling code in the linux (tm) kernel, Linux kernel documentation.
[50] J. Dean, L. A. Barroso, The tail at scale, Communications of the ACM 56 (2013) 74–80.
URL http://cacm.acm.org/magazines/2013/2/160173-the-tail-at-scale/fulltext
[51] T. Kaler, Y. He, S. Elnikety, Optimal reissue policies for reducing tail latency, in: Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures, ACM, 2017, pp. 195–206.
[52] K. Deierling, In modern datacenters, the latency tail wags the network dog, accessed: 2019-01-03.
URL https://www.nextplatform.com/2018/03/27/in-modern-datacenters-the-latency-tail-wags-the-network-dog
[53] C. J. Van Rijsbergen, The geometry of information retrieval, Cambridge University Press, 2004.
[54] C. Poultney, S. Chopra, Y. L. Cun, et al., Efficient learning of sparse representations with an energy-based model, in: Advances in neural information processing systems, 2007, pp. 1137–1144.
[55] M. A. Ranzato, Y.-L. Boureau, Y. LeCun, Sparse feature learning for deep belief networks, in: Proceedings of the 20th International Conference on Neural Information Processing Systems, NIPS'07, Curran Associates Inc., USA, 2007, pp. 1185–1192.
URL http://dl.acm.org/citation.cfm?id=2981562.2981711
[56] V. Nair, G. E. Hinton, Rectified linear units improve restricted boltzmann machines, in: Proceedings of the 27th international conference on machine learning (ICML-10), 2010, pp. 807–814.
[57] G. Alain, Y. Bengio, What regularized auto-encoders learn from the data-generating distribution, The Journal of Machine Learning Research 15 (1) (2014) 3563–3593.
[58] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, The Journal of Machine Learning Research 15 (1) (2014) 1929–1958.
[59] D. Kinga, J. B. Adam, A method for stochastic optimization, in: International Conference on Learning Representations (ICLR), Vol. 5, 2015.
[60] K. Yamanishi, J.-I. Takeuchi, G. Williams, P. Milne, On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms, Data Mining and Knowledge Discovery 8 (3) (2004) 275–300.
[61] M. Kristan, A. Leonardis, D. Skočaj, Multivariate online kernel density estimation with gaussian kernels, Pattern Recognition 44 (10-11) (2011) 2630–2642.
[62] L. Li, R. J. Hansman, R. Palacios, R. Welsch, Anomaly detection via a gaussian mixture model for flight operation and safety monitoring, Transportation Research Part C: Emerging Technologies 64 (2016) 45–57.
[63] H.-Q. Wang, Y.-N. Cai, G.-Y. Fu, M. Wu, Z.-H. Wei, Data-driven fault prediction and anomaly measurement for complex systems using support vector probability density estimation, Engineering Applications of Artificial Intelligence 67 (2018) 1–13.
[64] B. Schölkopf, R. C. Williamson, A. J. Smola, J. Shawe-Taylor, J. C. Platt, Support vector method for novelty detection, in: Advances in neural information processing systems, 2000, pp. 582–588.
[65] K. A. Heller, K. M. Svore, A. D. Keromytis, S. J. Stolfo, One class support vector machines for detecting anomalous windows registry accesses, in: Proc. of the workshop on Data Mining for Computer Security, Vol. 9, 2003.
[66] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., Scikit-learn: Machine learning in python, Journal of machine learning research 12 (Oct) (2011) 2825–2830.
[67] B. Hoyle, M. M. Rau, K. Paech, C. Bonnett, S. Seitz, J. Weller, Anomaly detection for machine learning redshifts applied

to sdss galaxies, Monthly Notices of the Royal Astronomical Society 452 (4) (2015) 4183–4194.
[68] K. M. Ting, F. T. Liu, Z. Zhou, Isolation forest, in: 2008 Eighth IEEE International Conference on Data Mining(ICDM), Vol. 00, 2008, pp. 413–422. doi:10.1109/ICDM.2008.17.
URL doi.ieeecomputersociety.org/10.1109/ICDM.2008.17
[69] Z. Ding, M. Fei, An anomaly detection approach based on isolation forest algorithm for streaming data using sliding window, IFAC Proceedings Volumes 46 (20) (2013) 12–17.
[70] H.-P. Kriegel, P. Kröger, E. Schubert, A. Zimek, Loop: Local outlier probabilities, in: Proceedings of the 18th ACM Conference on Information and Knowledge Management, CIKM '09, ACM, New York, NY, USA, 2009, pp. 1649–1652. doi:10.1145/1645953.1646195.
URL http://doi.acm.org/10.1145/1645953.1646195
[71] J. Tang, Z. Chen, A. W.-c. Fu, D. Cheung, A robust outlier detection scheme for large data sets, in: In 6th Pacific-Asia Conf. on Knowledge Discovery and Data Mining, Citeseer, 2001.
[72] M. Markou, S. Singh, Novelty detection: a reviewpart 1: statistical approaches, Signal processing 83 (12) (2003) 2481–2497.
[73] V. Hodge, J. Austin, A survey of outlier detection methodologies, Artificial intelligence review 22 (2) (2004) 85–126.
[74] F. Pedregosa, G. Varoquaux, A. Gramfort, et Al., Scikit-learn: Machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.