Anomaly Detection and Anticipation in High Performance Computing Systems

Anomaly Detection and Anticipation in High Performance Computing Systems

Andrea Borghesi, Martin Molan, Michela Milano, Andrea Bartolini

***Abstract*—In their quest towards Exascale, High Performance Computing (HPC) systems are rapidly becoming larger and more complex, together with the issues concerning their maintenance. Luckily, many current HPC systems are endowed with data monitoring infrastructures that characterize the system state, and whose data can be used to train Deep Learning (DL) anomaly detection models, a very popular research area. However, the lack of labels describing the state of the system is a wide-spread issue, as annotating data is a costly task, generally falling on human system administrators and thus does not scale toward exascale.**

In this work we investigate the possibility to extract labels from a service monitoring tool (Nagios) currently used by HPC system administrators to flag the nodes which undergo maintenance operations. This allows to automatically annotate data collected by a fine-grained monitoring infrastructure; this labelled data is then used to train and validate a DL model for anomaly detection. We conduct the experimental evaluation on a tier-0 production supercomputer hosted at CINECA, Bologna, Italy. The results reveal that the DL model can accurately detect the real failures, and, moreover, it can *predict* the insurgency of anomalies, by systematically anticipating the actual labels (i.e. the moment when system administrators realize when an anomalous event happened); the average advance time computed on historical traces is around 45 minutes. The proposed technology can be easily scaled toward exascale systems to easy their maintenance.

***Index Terms*—High Performance Computing, Anomaly Detection, Deep Learning**

I. Introduction

On the path toward Exascale computing, several challenges have been tackled. In 2008 US DARPA estimated that a feasible power envelope for the Exascale system would be 20MWatts, requiring 50GFlops/Watt of energy-efficiency (>100x improvement in energy-efficiency w.r.t. the 1st system in TOP500 in 2008) [1]. Today, Fukagu, the most powerful supercomputer worldwide (#1 system in the TOP500 list), comprises 159K nodes and consumes 30MWatts of IT power for 442PFlops with an energy-efficiency of 15GFlops/Watt [2]. Today's most energy-efficient supercomputer (#1 system in the GREEN500 list) achieves 26GFlops/Watt thanks to its heterogeneous design (NVIDIA DGX A100 GPUs and AMD EPYC CPUs) [3]. It is expected that the first generation of Exascale supercomputers will exceed 2008's estimated budget and will consume 30-40MWatts of power consumption for the IT only [4]. This will cause higher infrastructure and operational management complexities [4]. Fugaku has built-in mechanisms based on jobs' power prediction to avoid full-swing heat-load changes, which the chillers cannot follow [5].

Orthogonal to the energy-efficiency challenge, fault tolerance is of utmost importance [1]. Checkpoint methods can be used to tolerate fail-stop faults, but the introduced overhead increases with the node count and failure probability [1]. Fatter compute nodes can be adopted to limit Exascale systems' node count, or a lightweight checkpoint solution is needed [1]. Authors of [6] report an increase of jobs failure rate from 0.8% to 16.2% when doubling the job's size (from 10K to 20K nodes). If failure predictors are available, the checkpoint cost can be decreased [7].

Anomaly prediction methods based on Deep and Machine Learning (DL/ML) approaches can be trained using historical data sets. However, failure events are scarce in supercomputers, like in many other industrial plants. Approaches in the state-of-the-art bypass this method by either deploying an artificial fault injection framework in compute nodes [8]–[10] or using unsupervised or semi-supervised methods trained on the normal node state [11], [12]. None of these approaches have been validated with real faults in an entire production supercomputer.

As the size of supercomputing systems approaches the exascale, it is common to adopt Operational Data measurement, collection and Analysis (ODA) frameworks [13] to continuously monitor system information data (mostly in the form of multivariate time series data), such as data coming from physical sensors' telemetry (temperature, power), micro-architectural events (IPC, cache misses), data coming from the computing resources and facility [13]–[15]. These do not contain the records of node's and system failure events. However, the best practice (default operation) of the HPC system or a data center relies on the use of tools for event monitoring, software service and node status reporting [16]. These tools automatically warn system administrators about critical conditions, which can then be verified by manually inspection; a widespread tool used for this task is Nagios [16]. If the inspection confirms the critical state, the compute node has to be "drained" from the production; this failure (downtime) condition is recorded back into Nagios for post-mortem analysis.

Can we use Nagios messages as labels indicating that

Andrea Borghesi, Martin Molan, Michela Milano and Andrea Bartolini are with the University of Bologna, DISI and DEI Department, Italy.
E-mail: andrea.borghesi3@unibo.it, martin.molan2@unibo.it, a.bartolini@unibo.it, michela.milano@unibo.it

Andrea Borghesi, Michela Milano and Andrea Bartolini are also with the Alma Mater Research Center for Human-Centered Artificial Intelligence, Bologna, Italy

a supercomputing node is in a normal or faulty state, and use it to train DL models for fault prediction in a supervised fashion, as it is commonly done in state-of-the-art approaches on synthetic or simulated data? Secondly, *can we detect, or even better predict, these failures based on operational and facility monitored metrics?*

A. Contributions

To answer these research questions, in this manuscript we extend Examon, a state-of-the-art ODA framework [13], [15], deployed on the CINECA[1] data center, to integrate Nagios [16] monitored events. We are the first in exploring the possibility to use Nagios as an annotation tool (to provide normal and faulty state labels).

We demonstrate that the labels are indeed useful, but not enough: (i) we first show how pure supervised methods [8], [10] mostly learn trivial correlations (i.e. idleness equals to failure) and have no anticipation capability; (ii) then we show that using only semi-supervised method [11], [12] leads to suboptimal performance (high number of false positives); (iii) finally, we propose a new approach which combines a semi-supervised and a supervised model which is both accurate (with an F-score around 0.86) and can anticipate anomalies (around 1 hour before the anomaly is registered by system administrator.

We believe this methodology has high potential to increase the robustness and maintainability of future exascale supercomputers.

II. Related Works

Anomaly detection is a topic of interest in many different industries. One of the first applications of anomaly detection models were credit card fraud detection models in financial industries [17]. In recent years anomaly detection (and associated predictive maintenance) have become relevant in manufacturing industry [18], IT security [19] and even in complex physics experiments [20]. On the field of HPC preliminary explorations towards the creation of model for fault detection have been made. In the context of HPC system we consider anomalies in as periods of sub-optimal operation or faults that result in failed or erroneously completed jobs. Despite several possible failure mitigation [21] and fault tolerance strategies [22], anomalies in HPC systems still result in significant loss of available compute time [23]. The negative impact of failures and anomalous conditions in HPC machines is already significant and the problem will be only exacerbated by the quest towards Exascale, with the increasing number and heterogeneity of hardware (HW) components and software (SW) complexity [24], [25]. Supercomputer anomalies are rare events and as such they fall in the field of anomaly detection. Anomaly detection can be framed as an extreme case of unbalanced supervised learning problem [26], as the vast majority of data generated by real supercomputers is, by definition, normal. Due to the extreme unbalance classical supervised ML approaches might not give optimal results [19].

Broadly speaking, approaches for anomaly detection can be divided into two categories: 1) techniques that try to modify the data and 2) techniques that address the learning problem with a custom algorithm. Data manipulation approaches aim to mitigate the unbalance of the data sets either by undersampling the majority class and oversampling the minority class [27], or artificially creating new representatives of the minority class [28]. Custom algorithms that differ from classical supervised learning methods aim to learn the characteristics of the majority class and then recognize the anomalies as deviations from these learned characteristics [26]. In the HPC context, the majority of researchers have focused on specific techniques and data manipulation approaches has not been deeply studied, yet. This is due to another big challenge in HPC systems: having labeled data, that is having historical data sets where each data point is classified as normal or faulty state. Such data is a fundamental pre-requisite for the supervised ML techniques. Annotating data has a significant costs in terms of effort required to system administrators and facility managers

To mitigate this issue, often researchers "inject" anomalies in the supercomputers use as target and use case (see for instance Netti et al. [29]). In this way they can exact information about the state of the monitored system, thus having labelled data to train ML models. These ML models can then be directly trained on the collected data or after having applied a data processing step to obtain more synthetic but expressive features. Tuncer et al. [9], [10] deal with the problem of diagnosing performance variations in HPC systems. The authors train different ML algorithms to classify the behaviour of the supercomputer using the gathered data. In a similar fashion, Netti et al. [8], [29] propose a model based on Random Forest to classify different types of faults that can happen in a HPC node. Both Tuncer and Netti work on historical data sets collected from real supercomputers but they consider synthetic anomalies artificially injected in the HPC system.

In recent years, a different method to cope with label scarcity was proposed, namely relying on the knowledge that the vast majority of the data gathered on a supercomputer represent a normal operating condition. For instance, Borghesi et al. [11], [12], [30] propose the usage autoencoders to learn the characteristic behaviour of a supercomputer in a healthy state; they use a particular type of DL neural network called *autoencoder*. The learning task of the autoencoder is to reconstruct the initial input sequence; the internal representation of the autoencoder is constrained in such a way that the model cannot simply learn an identity function (e.g., by projecting the input data on a lower-dimension latent space). These trained networks are then use to classify between normal and anomalous points in incoming data streams. Anomaly detection approaches can also use the information about the failure proximity (similar HW architecture, physical or

[1]The largest Italian computing center, residing in Bologna

temporal locality, etc.) to increase the accuracy of anomaly detection, as discussed by Ghiasvand et al. [31].

The important difference between existing work and the work in this paper is the use of data from real HPC production. Existing work such as [8]–[11] uses anomaly injection to *artificially* create faults that are then approximated with a ML model. As such, these models were not tested on production systems. An ML approach capable of recognising the dynamics and characteristics of injected faults might not be able to recognise the dynamic of *actual failures* – the dynamics of failures might be more complex and more difficult to model than those of injected anomalies. Preliminary research works addressing anomaly detection and fault prediction using real anomalies have been proposed, but they tend to focus on the availability of a single (HW or SW) component of the system and not on the availability of the entire computing nodes. Ostruckow et al. [32] analyze the (specific) failures of GPU processors, Boixaderas et al. [23] aim at predicting the memory (DRAM) failures, Di et al. [33] detect silent data corruption, Groves et al. [34] predict sub-optimal operation due to memory contention. These approaches focus on specific HW components, but in the age of Exascale, with larger systems, more components and higher costs, such partial detection and monitoring systems should be combined and enhanced with supervised and unsupervised holistic anomaly detection models [14], [24]. To the best of our knowledge, the approach presented in this work is the only one that combines the holistic approach to system availability (overall system availability as opposed to component availability) with training and validating models on real production data (as opposed to generated anomalies).

III. ExaMon

ExaMon is a holistic framework for HPC facility monitoring and maintenance [35], designed for very large scale computing systems, such as supercomputers. It has been developed for the Exascale, thus stressing the capability to handle big data from many heterogeneous sources. At the lowest level, there are *collector* components to read the data from several sensors scattered across the system and deliver them, in a standardized format, to the upper layers of the stack. There are collectors with direct access to HW resources and collectors that sample data from other applications, such as batch schedulers and SW diagnostic tools. ExaMon has been deployed on CINECA machines since 2017 [15].

The infrastructure is built using the MQTT protocol[2]. MQTT implements the publish-subscribe messaging pattern and requires three different agents: (i) the publisher, that sends data on a specific topic; (ii) the subscriber, that needs specific data, so it subscribes to the appropriate topic; (iii) the broker, which (a) receives data from publishers, (b) makes topics available to subscribers, (c) delivers data to subscribers. When a publisher agent sends some data with a certain topic as a protocol parameter, it is created and made available by the broker. Any subscriber to that topic will receive the associated data when published by the broker. In ExaMon collector agents have the role of publishers.

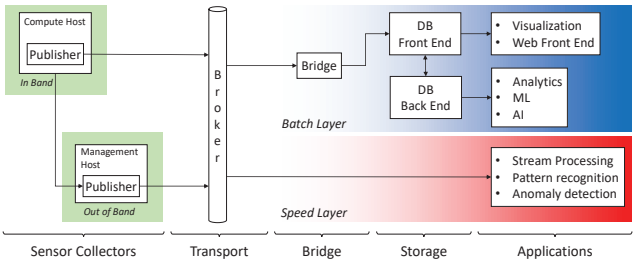


Fig. 1: ExaMon Architecture

The collected metrics are stored on a distributed and scalable time series database (DB), KairosDB [36], built on top of a NoSQL DB, Apache Cassandra [37] as back-end. A specific MQTT subscriber (MQTT2Kairos) is implemented to bridge the MQTT protocol and the KairosDB data insertion mechanism. The bridge leverages the MQTT topic structure to compose the KairosDB insertion statement automatically. This gives a twofold advantage: first, it lowers the computational overhead of the bridge since it is reduced to a string parsing operation per message; and secondly, it makes it easy to form the DB query starting only from the knowledge of the matching MQTT topic. The sampling rate of the metrics measured by ExaMon can vary among the different metrics, depending on the underlying sensors. However, ExaMon aggregates the data in 5-seconds windows; namely, a data point with time-stamp t represents the average of measurements sampled over the $[t-5s, t]$ window. Every five seconds a MQTT packet is built and sent to the broker; ExaMon low-level plugins are in charge with aggregating sensors measurements with higher sampling rates.

At the lowest level ExaMon is based on sensor collectors which retrieve the data from the sources and share it, trough the transport layer, to the other components of the framework. Each data source has its own specific sensor, tailored on its peculiarities. ExaMon collects physical data measured with HW sensors and program counters, e.g. HW components as cores PMU, IPMI, GPU, I2C, PMBUS. The collected data cover a wide ranges of sources, for instance, CPU load of all the cores in the supercomputing nodes, CPU clock, instructions per second, memory accesses (bytes written and read), fan speed, the temperature of the room hosting the system racks, power consumption (at different levels), etc. There are a few hundreds of metrics collected on each computing nodes, plus dozens covering the racks and rooms. This large amount of data allows to have a very fine-grained overview of the whole system, which is required to correctly characterize the supercomputer status and to discern between normal states and failures.

ExaMon collects data generated by the job dispatcher

[2]http://docs.oasis-open.org/mqtt/mqtt/v3

as well, in particular Slurm [38]. The job dispatcher in a supercomputer is the SW component that handles users requests and decides when each job should start and which resources to allocate; to gather information about job requests we created a plugin which gathers the information from Slurm and sends it to the data collection backbone via MQTT. The collected data regards the job request (job id, job name, job user, job partition/queue) and the requested resources (number of requested nodes, requested cores, requested GPUs and/or other HW accelerators, requested memory, requested wall-time). Another group of information pertains the job actual execution: submission time, execution start time, end time, the set of nodes actually used along with the cores.

A. Nagios

In addition to data coming from physical sensors, ExaMon also collects information related to service monitoring and node status using Nagios [16]; by using Nagios and its alarm generators, system administrators are warned about potentially critical conditions, which are then manually verified – if the alarm was real, the involved computing nodes have to be removed from production ("drained"). When nodes are marked as not in production by system administrators their different state is registered in Nagios as well – technically, they are put in a "DOWN+DRAIN" state. The data sampling frequency is 15 minutes, as determined by Nagios monitoring functionalities. In most HPC systems service-reporting tools are decoupled from physical monitoring infrastructure, but thanks to ExaMon we can merge them and (hopefully) reap the benefits.

Nagios is composed by a core part which monitors and visualizes critical IT infrastructure components. It also provides alerts and historical logs of variations in the state of each monitored components. To monitor the state of a given SW and HW component Nagios can be extended with predefined and custom plugins, which reads metrics related from target components and based on specific rules defines its state as "Normal", "Warning" and "Critical". Only state transition events are recorded in the logs. These values are periodically sent as MQTT messages to ExaMonto be stored as additional metrics. System administrators use Nagios to monitor the state of CINECA HPC machines, using a modular customization adapted to the large scale[3]. Nagios provides a central view of the system status. Different dashboards provide access to monitoring information and views provide users with quick access to useful information; it collects the results of active and/or passive checks on different hosts and related services. Alerts can be sent to management staff, which are then handled by automated scripts that mark the node status depending on the severity of the alert.

The critical observation is that the inspection done by system administrators through Nagios can be exploited to distinguish between nodes in normal states and nodes in critical conditions – "DOWN+DRAIN" state indicates anomalous conditions while other states indicate normal behaviour[4]. This information can be framed as labels describing the status of the supercomputer, thus providing (for free) the automated annotation operation which is lacking in modern HPC systems. The goal of this paper is to train a ML model with the annotated data gathered by ExaMon, and to predict critical HPC node failures that are recorded as "DOWN+DRAIN" events (in the rest of the paper when we refer to *system failures* we mean with "DOWN+DRAIN" events).

IV. Detecting Anomalies with Nagios

In this paper we want to investigate the possibility to use Nagios-provided labels to train a DL model for anomaly detection in a HPC system. It is important to observe that using Nagios-provided labels is a promising direction but comes with downsides. In particular, a HPC node is put in "DOWN+DRAIN" state *only when a system administrator notices the issue*; then, the node is detached from production. This procedure has a twofold implication: i) there is an implicit delay between the insurgence of an anomalous situation and the corresponding label, and ii) labels indicating anomalies tend to be associated to nodes in idle state, as during the maintenance performed by system administrator to identify and fix the source of the issue no new jobs are submitted on the node (excluding diagnostic tools with lower impact on a computing node compared to the typical HPC workload).

This situation complicates the creation of an automated model for anomaly detection. We do not want a DL model that simply "learns" the correspondence between failure state and idleness. Concerning the evaluation of trained models, we cannot simply consider standard classification metrics such as the accuracy. Figure 2 graphically details the issue; the time is on the x-axis while the y-axis represent the anomaly state identified via Nagios (0 indicates normal state, 1 indicates failure). Let us focus on the second failure in the figure. We can identify six different phases: ① corresponds to the supercomputing node in normal state; in phase ② the label is still set on 0 as the system administrator has not noticed the failure, yet; in ③ the node is recognized as faulty and removed from production – running jobs are still completing or handling termination signals, hence the node is not idle yet; ④ represents the maintenance period, when the node is mostly in idle state; in ⑤ the administrator runs the final check (after having solved the issue) and prepares the node for re-insertion in production; finally, in phase ⑥ the Nagios flag is set again to normal state and the node is made available – however, for some time it can still be in an idle state, as new jobs need to be submitted for the node resources to be allocated. Ideally, an automated

[3]Full details can be found here https://prace-ri.eu/wp-content/uploads/Design_Development_and_Improvement_of_Nagios_System_Monitoring_for_Large_Clusters.pdf

[4]In this preliminary analysis we focus on binary classification (e.g., normal versus abnormal situations); in future works, the different states recorded by Nagios can be distinguished with more sophisticated, multi-class approaches.

method for anomaly detection should have high accuracy in all phases except the second one, as in this phase a faulty situation is already ongoing, albeit not detected yet by system administrators – having a large number of false positives here is actually a good outcome (it means that the failure can be *anticipated*). Additionally, phase ③, ⑤, and ⑥ might complicate the task for the ML model, as they require the model not to simply learn that fault equals to idleness.

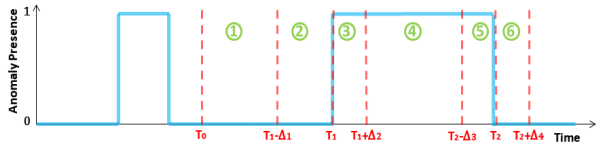


Fig. 2: Nagios anomaly signal; the light blue line indicates the presence of faults (value 1)

A. The Proposed Approach

Furthermore, approaches from the literature (e.g., [8], [10], [12]) which rely on fault injection assume to know the exact moment when the failure was injected. In this works we show that in a real context it is impossible to know *exactly when a failure event starts*, but we can know the *moment when someone has notified us the problem*, after having noticed it. The experimental results discussed in Sec. V-B demonstrate that techniques designed with artificial failures struggle when dealing with real faults – in short, fully supervised methods fail to *anticipate* anomalies (no false positives in phase ②) and mostly learn the idle-failure correlation.

However, not using the information provided by Nagios labels and adopting a semi-supervised approach (as our previous works [12]) proved to lead to inaccurate detection rates (with very high number of false positive, discussed in Sec. V-C). Hence, the core of the approach proposed in this paper is the union of two DL models, namely a 1) *semi-supervised autoencoder deep neural network (DNN)* and 2) a *supervised neural network* composed by an autoencoder (distinct from the semi-supervised one) and a series of classification layers. The two DL approaches (semi-supervised and supervised) are trained separately. Their output is combined only at inference time, i.e., when making a prediction on new data. Both models produce an anomaly signal, classifying data points as representing normal (anomaly signal equal to 0) or faulty states (signal equal to 1) of the supercomputing nodes. The predictions of the two models are combined to generate the final signal: if any model raises the anomaly flag, then the data point is classified as anomalous. The anomaly signals of the two models are thus combined in a logic-OR fashion: if both signals have value 0 then the new point is classified as normal, in all other cases is classified as anomalous. For the moment, we are training a different couple of models (supervised and semi-supervised) for each node in the supercomputer; in future works we will explore models capable of handling multiple nodes (e.g., system- or rack-wide models).

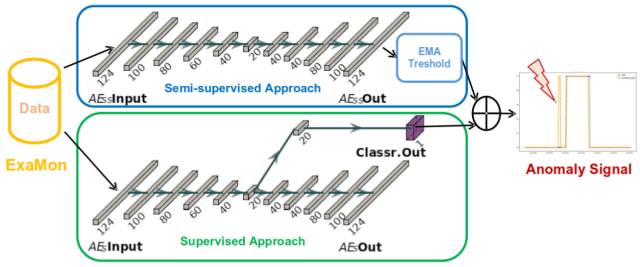


Fig. 3: Combined Approach Scheme

The overall scheme of the approach is displayed in Figure 3. The number of neurons of each layer is reported as well; all layers belonging to the autoencoders have ReLU activation function, the final classifier layer has a softmax as activation function. The autoencoder networks used for the semi-supervised approach and for the supervised one have the same topology (number of layers, neurons, etc) but are trained separately (using only normal data in the semi-supervised case, both normal and anomalous in the supervised one). For the sake of simplicity the figure does not distinguish between the different types of data that are fed to the semi-supervised and the supervised approaches, nor does it show the convolution operation applied to the input data for the semi-supervised autoencoder.

1) The Semi-supervised Approach: is composed of an autoencoder DNN [39] trained *using only normal data*; in this way the autoencoder learns the correct behaviour of a node (the idea was originally proposed in the HPC context by a previous work [11]). Then, the network can be used for anomaly detection by observing its reconstruction error computed on new data: if it is greater than a threshold, the new sample is classified as anomalous[5], normal otherwise. The selection of the threshold is a non-trivial issue; in previous works it was computed using a sub-set of data and by exploring different values through grid search. The reconstruction error is computed as the average error over all the features (by construction, the autoencoder has the same number of features both in input and in output). In this work we extend the previous model in two significant directions: i) pre-processing the data through a convolutional step and ii) modifying the threshold computation. The convolution is used to "smooth" the raw data and to more explicitly consider the temporal dynamics involved in the computing nodes behaviour; we employed a convolutional 1-dimensional layer with a filter of a size corresponding to 2 hours. Concerning the threshold, we did not adopt a fixed one but we employed Exponential Moving Average (EMA), obtaining a variable threshold which follows the error trend in previous time steps, giving more weight to more recent observations. For a series of

[5]High error means that the autoencoder does not "recognize" the data.

errors E EMA may be calculated as:

$$EMA_t = \begin{cases} E_1, & \text{if } t = 1 \\ \alpha E_t + (1-\alpha) EMA_{t-1}, & \text{if } t > 1 \end{cases} \quad (1)$$

, where E_t is the reconstruction error at time t, EMA_t is the value of EMA at time t, and $\alpha \in [0, 1]$ is a parameter that weighs the importance of more recent time steps[6]. The simplest way to use an EMA threshold is to consider anomalous the data point at time step t_i if the corresponding reconstruction error computed by the autoencoder E_i is larger than the EMA error. However, a preliminary analysis revealed that this simple method was too sensitive and generated a very large number of false positives; the anomaly signal was "triggered" far too often. We then opted for a less sensitive threshold: a data point is classified as anomalous only if the reconstruction error is larger than the EMA threshold plus a δ value, in order not to have a signal which overreacts to any non-significant oscillation. In particular, we compute the δ as the average reconstruction error over the previous 25 minutes. The 2-hours convolutional filter and the 25-minutes period were obtained during a non-exhaustive, manual search, conducted in a preliminary empirical evaluation. We will definitely explore more systematic approach in the future (e.g., Bayesian Optimization).

2) The Supervised Approach: consists of an undercomplete autoencoder network[7] for feature extraction; this autoencoder is pre-trained in an unsupervised manner (minimizing the reconstruction error plus a regularization term); in this case we feed the autoencoder with all data points in the training set, including both normal and anomalous points. On top of the autoencoder two classification layers have been added; these latter layers are trained using the labels (provided by Nagios) by minimizing the categorical cross-entropy. The training happens in two phases: first, the unsupervised autoencoder is trained, then its weights are fixed and only the encoder is used to train the classifier layers in the supervised manner. At inference time, the anomaly signal is generated by feeding new data to the encoder, obtaining a compressed representation, and then the latent representation is passed to the classifier, which provides a real number as output (i.e. the probability of belonging either to normal or faulty class); if the output is higher than a threshold the point is classified as failure – in this paper we employ a threshold equal to 0.2[8]. This threshold and the other hyperparameters describing both the supervised and the semi-supervised models[9] were manually fine-tuned during a preliminary empirical evaluation.

[6]Higher α discounts older observations faster.

[7]"Undercomplete" means that the latent representation learned by the network has a lower dimension compared to the input features

[8]Using higher threshold would lead to fewer false positives, but in our case we do not want to minimize them.

[9]Convolution filter size, EMA α, number of layers, etc.

V. Experimental Evaluation

This section presents the results of the experimental evaluation. We start by describing the data set used for training and testing the DL models. Then, we describe the preliminary results obtained using a fully supervised approach, observing its limitations; afterwards, a semi-supervised approach not using the labels is described, showing promising qualities but significant drawbacks. Finally, the results obtained by combining the semi-supervised method with the supervised one are discussed in Sec. V-D. A simplified version of the code used to perform the experimental evaluation has been published[10].

As noted earlier (Sec. IV) we know that the labels provided by Nagios are delayed with respect to the actual insurgence of the fault; for this reason we cannot measure the quality of the approaches discussed in the section (and following ones) merely looking at standard metrics such as accuracy and F-score, nor the confusion matrix is sufficient. Instead, we will discriminate between the detection errors made just before and after the faults indicated by the labels (we look two hours before and two hours after the labelled anomaly); these are the false positives happening in phases ② and ⑥ depicted in Fig. 2. Additionally, we are interested in understanding whether the DL model can *anticipate* the anomaly labeled through Nagios; we compute the anticipation observing the false positives happening before the actual label. We assume that a fault is anticipated if the approach generates three[11] consecutive false positives (corresponding to 15 minutes of uninterrupted anomaly signal); non-consecutive false positives do not contribute to the anticipation – however, they are clearly not discarded from the computation of the overall evaluation metrics (F-score, etc.).

A. The Data Set

In order to test our approach we use historical data collected from the tier-0 production Marconi supercomputer [40], hosted at CINECA, Bologna. Marconi has been upgraded between 2016 and 2020 in three main phases; the current system is composed by 3188 nodes, each equipped with two 24-cores Intel Xeon 8160 (SkyLake) processors and 196GB of RAM memory. The total peak performance of the overall system is around 20PFlops, with 17PB available storage space. ExaMon has been deployed as well on the recent upgrade Marconi100, a supercomputer based on IBM Power 9 chips; Marconi100 ranked ninth in the June 2020 Top500 list [3], with an achieved peak performance of 21.6 petaflops (performance per node around 32 TFlops). Marconi100 is composed by 980 computing nodes, each one with two 16-cores IBM POWER9 and 4 nVidia Volta 100; the total RAM per node is 256GB. The storage space for the whole system is 8 petabytes. ExaMon has been deployed on Marconi since the first semester of 2019.

[10]https://github.com/MolanM/Anomaly-Detection-and-Anticipation-in-HighPerformance-Computing-Systems

[11]Value empirically chosen.

TABLE I: Data set overview: number of normal and anomalous data points (average over all nodes), total number of data points (Tot.) and percentage of anomalous samples w.r.t. the total number (% Anom.)

Month	# Normal	# Anomalous	Tot.	% Anom.
Jan. 2020	5678.60	22.50	5701.10	0.39
May 2020	6984.30	194.70	7179.00	2.71
Jan. & May	12658.25	216.20	12874.45	1.68

The data set combines sensor measurements coming from a variety of HW sensors and program counters, information from the job scheduler (SLURM), and reports on the system availability and status updates collected by Nagios. We do not use the raw data coming from ExaMon, with its 5-seconds sampling rate, but we rather employ aggregated metrics, namely the average value and the standard deviation computed over 5-minutes interval. This was done for a two-fold purpose: 1) aggregating data along the time axis with the "mean" operator allows to implicitly consider the temporal dynamics involved in the evolution of the supercomputing nodes; 2) as Nagios provides status updates every 15 minutes (see Sec. III-A), considering the raw 5-seconds intervals would create a lot of noise and would not provide useful information, given the significant difference in the raw data and labeling frequencies.

For this analysis we selected a random subset of twenty nodes from Marconi. The data is divided in two groups belonging to two different months, January and May 2020; each period is then split in 20 data sets, each one corresponding to one of the selected nodes. As described earlier we use Nagios data to annotate the data, meaning that each time stamp was labelled using the information about the status of the computing node. The label (target) can assume two values: 0 describing normal operation and 1 describing anomalous states ("DOWN+DRAIN" event as recorded by Nagios). The data set has been made public[12]. Table I provides a brief overview of the number of normal and anomalous data points in the data set (averaged over all 20 nodes); as expected in a production HPC system, the failure events are extremely rare, consisting in less than 2% of the total number of data points. For each node we selected 124 metrics, a subset of the metrics collected on Marconi nodes by EXAMON (we discarded metrics with missing values). Hence, 122 is the size of the input layers for the autoencoders described in Sec. IV-A; the input are real numbers. Using one of Marconi nodes for training and testing the composite model (the combination of semi-supervised and supervised approaches), the training time on one month of data is between 20 and 30 seconds (for thirty training epochs), while the he inference time for a single data sample is between 13-16ms.

B. Supervised Approach Results

We start by considering the results of the pure supervised approach. This is the kind of approach proposed by the current state-of-the-art for fault classification in supercomputers (e.g., see [10], [41]). However, the works in the literature consider artificially injected faults and not real production anomalies; they also focus on "reliable" labels which reflect the actual underlying change in the target systems. We are instead interested in anomalies in a production HPC machine, and we want to investigate whether Nagios-based annotation can be used to train automated anomaly detection models; we are not proposing a new method for fault detection. Hence, we report the results obtained applying the supervised NN described in Sec. IV-A and another supervised method, namely Random Forest (RF, [42]), which in previous experiment from the literature proved to be the most accurate. For each computing node we trained a different RF and supervised NN; 70% of data from the node was used for training the remaining part for testing.

The RFs were implemented using *scikit-learn* Python module[13], while the DNNs were implemented using TensorFlow[14]. Table II reports the result of the comparison between the two supervised models. RF indicates the Random Forest[15]; "AE + Classr." indicates the DNN composed by an autoencoder plus the classification layers. The meanings of the columns are the following: *TP*, True Positives; *TN*, True Negatives; *FN*, False Negatives; *FP-Pre*, False Positives in the period preceding the Nagios-annotated label (two hours earlier); *FP-Post*, False Positives in the two hours following the cessation of the failure according to Nagios; *FP-Rnd*, False Positives randomly picked over the whole test set; *FP*, total number of False Positives (FP = *FP-Pre*+*FP-Post*+*FP-Rnd*); *TNR*, True Negative Rate ($TNR = TN/N$, where N is the number of real negative cases); *TPR*, True Positive Rate (also known as recall, $TPR = TP/P$, where P is the number of real positive cases); *Prec.*, Precision ($Prec. = TP/(TP+FP)$); *FNR*, False Negative Rate ($FNR = FN/P$); *F-score* is the harmonic mean of precision and recall – values close to one indicate greater detection accuracy[16]. The table reports the average results computed over the 20-nodes subset.

The results are very clear: both models were able to correctly learn to distinguish between normal states and anomalous ones, as highlighted by the very high values of precision, recall, and F-score. The accuracy number are on par with those reported by the state-of-the-art [10], [41], a significant first result for real failures happening on production supercomputing nodes. This is an important and promising step towards the adoption of Nagios as an annotation method. However, we must notice a partial limitation: the alarm signal computed by supervised models do not anticipate the labels and this suggests that the strong accuracy results are partially due to the fact that nodes in "DOWN+DRAIN" state have a significantly different behaviour compared to those in normal operating conditions (e.g., high level of idleness and very different

[12]https://doi.org/10.5281/zenodo.4537849

[13]scikit-learn, https://scikit-learn.org/stable/index.html

[14]https://www.tensorflow.org/

[15]We adopted the default parameters of the RF implementation provided by scikit-learn

[16]The F-score is defined as $F - score = 2 \cdot \frac{precision \cdot recall}{precision + recall}$

TABLE II: Comparison of the experimental results obtained with the supervised methods for anomaly detection on Marconi supercomputer. The results obtained on two months of data collected are reported; all values are average computed over twenty different computing nodes. Jan. & May rows are obtained by summing the two periods.

Month	Method	TP	TN	FN	FP-Pre	FP-Post	FP-Rnd	FP	TNR	TPR	Prec.	FNR	F-score
Jan. 2020	RF	20.40	5675.50	2.20	1.10	1.20	0.40	2.70	1.00	0.90	0.88	0.10	0.89
Jan. 2020	AE+Classr.	19.70	5669.60	2.80	1.50	4.70	2.90	9.10	1.00	0.88	0.68	0.12	0.77
May 2020	RF	191.60	6978.90	3.20	2.00	2.00	0.00	4.00	1.00	0.98	0.98	0.02	0.98
May 2020	AE+Classr.	186.40	6932.30	8.30	10.00	9.60	32.50	52.10	0.99	0.96	0.78	0.04	0.86
Jan. & May	RF	212.00	12654.40	5.40	3.10	3.20	0.40	6.70	1.00	0.98	0.97	0.02	0.97
Jan. & May	AE+Classr.	206.10	12601.90	11.10	11.50	14.30	35.40	61.20	1.00	0.95	0.77	0.05	0.85

workload). This was confirmed by looking at the features which govern the RF classifier, as the most important ones (for every node used in the experiment) were the average core load in the previous 5 and 10 minutes. In practice, an high accuracy does not entail concrete benefits in terms of forecasting failures in the HPC nodes. This is especially true for the RF model, while the slightly higher number of false positives obtained with the DNN reveals that this method is less focused on the idleness. For this reason, in Sec. V-D we opted for the DNN, as part of the composite approach. Another drawback of the supervised models the lack of anticipatory capability, as they are very good at detecting the difference between normal states and failures, but cannot forecast the insurgence of anomalies.

As the data set is extremely imbalanced (see Tab. I) classification models could struggle, though this was not the case. Nevertheless, we performed a preliminary experiment to address this issue, namely we employed a Python library[17] to pre-process the data via undersampling and oversampling techniques. The empirical results were not significant (e.g., very marginal differences in all the evaluation metrics), as the supervised methods were already sufficiently accurate.

C. Semi-supervised Approach Results

As we saw in the previous section, supervised methods have high accuracy but little utility in practice. We turn now to a semi-supervised approach, as previous works highlighted its potential [11], [12], [30]. In this case we employ the autoencoder DNN described in Sec. IV-A; we conduct the experiment on the same nodes used previously and we create an autoencoder model for each node – now only normal data is used for training the network, leaving out part of the normal points and all the anomalous ones for testing purposes. Table III reports the result obtained with the semi-supervised approach, using the same format which described the supervised methods. In the semi-supervised case we show the results with two different models: *AE-Base* corresponds to the work of Borghesi et al. [11], [12] and consists of an autoencoder network plus a fixed threshold[18]; *AE-Conv.* is the new model introduced in Sec. IV-A, which extends the original autoencoder with a convolution operation for smoothing purposes and employs a variable threshold based on EMA for generating the anomaly signal – for the results in Tab. III we set $\alpha = 0.5$.

We report the same evaluation metrics used for the supervised methods for the sake of comparison, as we expect the performance of the semi-supervised approaches to be worse. Indeed, we can notice a steep decrease in accuracy compared with the supervised approaches, especially for the autoencoder with fixed threshold, which has a very low accuracy rate due to the high number of false positives; moreover, these false positives are also evenly spread along the whole test set. However, the situation greatly improves when we consider the improved semi-supervised model with convolution and EMA threshold. In this case the F-score significantly raises, together with the sharp decrease in the number of false positives (and the precision significantly increases as well). In addition, while the overall number of false positives diminishes, those located before failures actually augments, suggesting that this semi-supervised network might actually anticipate the labels generated by Nagios. The main issue is that the not all false positives before the failures are consecutive, and thus are not used to signal anomalies (see Sec. IV-A). This is the gap that we hope to cover by combining the semi-supervised DNN with the supervised model composed by the autoencoder plus classification layers.

An additional benefit of the autoencoder network is the possibility to "interpret" the prediction of the model in a natural way. In particular, one can look at the different reconstruction errors obtained for each feature taken as input and output of the autoencoder. The reconstruction error of an autoencoder with multivariate input and output[19] can be obtained in different ways. The reconstruction error is the difference between the output and the input; this difference is computed feature-wise and then averaged. Using the aggregated reconstruction error is useful for obtaining a unique and coherent anomaly signal, but we can also observe the error generated in reconstructing each feature. Features with higher reconstruction errors have more anomalous behaviours, hence the autoencoder struggle to reconstruct them. These features can then be interpreted as the "root causes" of the failure, or at least a very strong indicator of which components of the systems are behaving in the most anomalous way. This can be observed by looking at the heatmaps

[17] https://github.com/scikit-learn-contrib/imbalanced-learn

[18] The fixed threshold is computed as the 95-th percentile of the reconstruction errors computed in the training set; see [12] for a more detailed explanation concerning this choice.

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

TABLE III: Comparison of the experimental results obtained with the semi-supervised methods; average results over all nodes. *AE-Base* indicates the results with the autoencoder with fixed threshold; *AE-Conv.* the results with the autoencoder plus convolution and EMA variable threshold.

Month	Method	TP	TN	FN	FP-Pre	FP-Post	FP-Rnd	FP	TNR	TPR	Prec.	FNR	F-score
Jan. 2020	AE-Base	7.40	5489.00	15.10	4.89	12.52	182.15	199.56	0.96	0.33	0.04	0.67	0.06
Jan. 2020	AE-Conv.	19.70	5664.10	2.80	2.20	9.50	4.10	15.80	1.00	0.88	0.55	0.12	0.68
May 2020	AE-Base	47.90	6732.40	146.90	4.10	3.90	229.60	237.60	0.97	0.25	0.17	0.75	0.20
May 2020	AE-Conv.	186.40	6923.60	7.30	4.35	7.60	42.80	54.75	0.99	0.96	0.77	0.04	0.86
Jan. & May	AE-Base	55.30	12221.40	162.00	8.99	16.42	411.75	437.16	0.97	0.25	0.11	0.75	0.16
Jan. & May	AE-Conv.	206.10	12587.70	10.10	6.55	17.10	46.90	70.55	0.99	0.95	0.74	0.05	0.84

portraying the feature-wise reconstruction errors along the temporal line, as shown in Figure 4 for two distinct nodes of Marconi (dubbed "Node A" and "Node B" for simplicity). The x-axis represents the time while the y-axis shows the reconstruction errors for a subset of the features of the nodes (we selected the most representative ones). The time is divided in 5-minutes intervals, indicated by the indexes from 0 to 48; we are thus representing an overall period of 240 minutes (48 multiplied by 5). The heatmaps contains 4-hours period containing fault events on the nodes; the red markers on the top row indicate the anomalous periods. The magnitude of the reconstruction error is conveyed through the color intensity, with hues close to white signifying low errors and hue close to intense blue representing high reconstruction error.

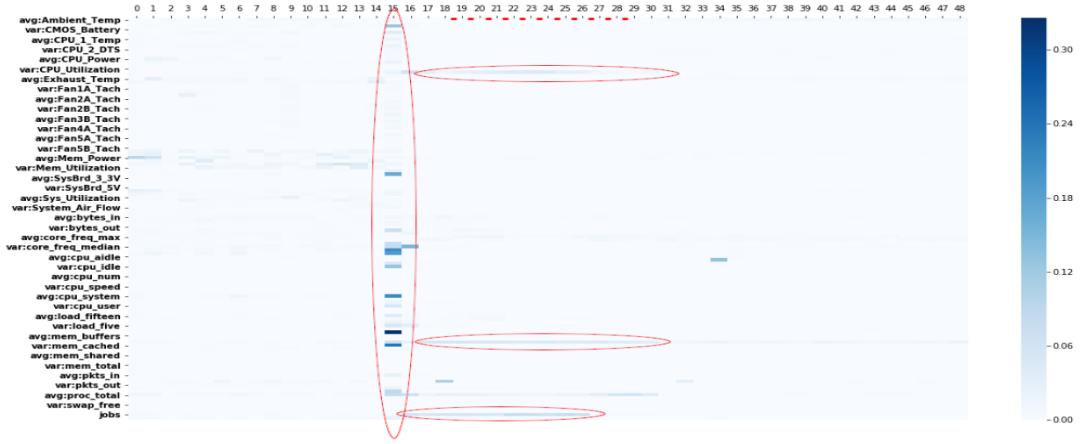
Let us start with Node A; the failure happened on January 16, 2020, from 9:05 to 11:00. We highlighted interesting areas of the map with red ellipses. One thing that can be quickly noted regards the reconstruction error during the anomalous periods, which is relatively higher for three specific features, "avg:mem_cached" (the amount of cached memory), "avg:DC_Energy" (the energy consumed by the entire node), and "jobs" (the number of jobs completed in the last five minutes). "avg" and "std" represent the average and standard deviation computed over the aggregation interval – see Sec. V-A. These features are enclosed by the three horizontally-aligned ellipses. The fact that the autoencoder fails to reconstruct these features during the anomalous period is probably due to the non-normal workload happening when the node is put in "DOWN+DRAIN" state (e.g., no new jobs are submitted). More interestingly, we can see that shortly before the anomaly (more precisely at time index 15, 15 minutes before the insurgency of the failure, time index 18) a significant spike of the reconstruction error involves a large number of features, in a marked contrast with the normal situation (the vertically-aligned ellipse). This strongly suggests that the semi-supervised autoencoder is aware that something strange is happening, or at least starkly different from the norm, hence it raises its anomaly signal, generating a false positive as the Nagios label is still set to normal state.

In the case of Node B we see similar patterns but also different ones; in this case the failure happened on May 27, 2020, from 15:20 to 16:00 (time steps [21, 25]). The heatmap displays a relatively faint but noticeable higher reconstruction error for many features before the anomaly insurgence (time steps [18-20]), especially for features involving metrics which describe the cooling components (fan speed, air flux, etc). However, there is a very high error for a particular feature, "avg:SysBrd_12V" (describing the system board voltage), which predates the rise of the error on other features. In this heatmap we can observe as well a markedly high reconstruction error around one hour and half after the failure event (second vertical ellipse), in particular for the feature "avg:bytes_in", which describes the number of bytes per second transferred from the network to the node. This generated an undesired false positive happening after the labelled anomaly; with the information currently stored by ExaMon and by system logs, we do not know whether this was an actual mistake of the semi-supervised model or an anomalous event of little importance and thus which escaped the Nagios annotation mechanism (that is, it did not cause significant disruption to request the system administrators intervention), but did not went undetected by the autoencoder – possibly, requiring its re-calibration to decrease its sensitivity. We plan to explore in more detail and validate this "explanation" capability and root-cause identification in future works, in strict collaboration with system administrators and facility managers.

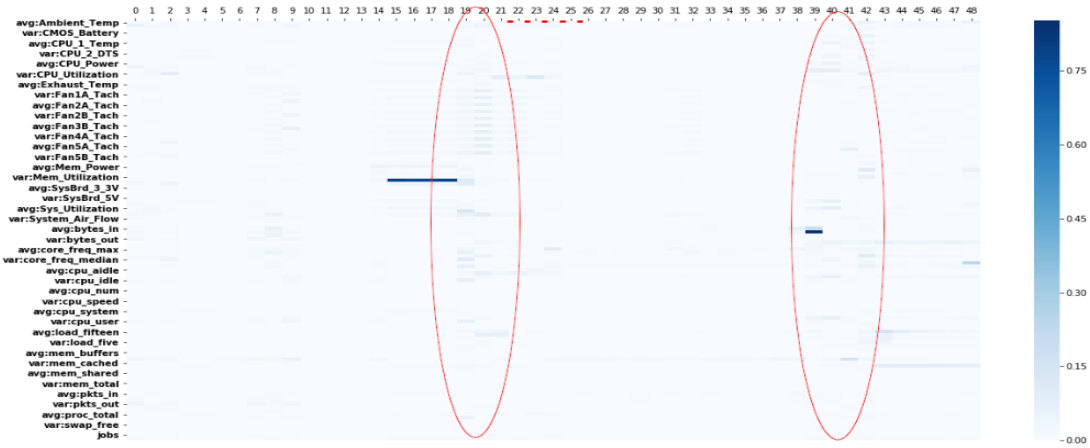
D. Combined Approach Results

In this section we explore the results of the approach which combines two DL models, the semi-supervised DNN and the supervised classifier built on top of the autoencoder. The idea is to merge the benefits of both worlds, possibly obtaining a method capable on anticipating the failures registered with Nagios. For instance, let us go back to Node A in Fig. 4. The semi-supervised approach noticed something "strange" 15 minutes before the actual anomaly (generating a false positive); however, no anomaly signal was generated as the false positive was not followed by two consecutive ones. If the supervised method was capable of producing false positives for the missing time periods, by merging the two models we could anticipate the failure as registered by Nagios by 15 minutes. As previously mentioned (Sec. IV-A), we merge the supervised and semi-supervised approaches by combining their anomaly signals in a logic-OR fashion.

Table IV reports the average results over all nodes, in a similar manner to the previous tables with aggregate results, but with two differences. First, there are no different models to be compared but we rather consider the



(a) Node A



(b) Node B

Fig. 4: Heatmaps obtained with the semi-supervised approach. The x-axis represents the time; the reconstruction error for a subset of the features is displayed along the y-axis (more intense blue indicates greater errors). Red markers on the top row represents failures.

same technique (that is, the combination of the supervised and semi-supervised models) with four different α values, where α specifies how much weight is given to most recent examples in the computation of the EMA dynamic threshold (see Eq. 1) used for the semi-supervised approach. We explore the α parameter as the EMA threshold is a novel extension compared to the previous semi-supervised NN; at this stage, we are not interested (yet) in exploring in depth all the possible values for the hyperparameters of the DL models[20] but we want to show the potential of the approach – future works will deal with the fine-tuning. The second difference compared to previous tables is that we do not show the True Negative Rate, as it is not so informative, but we added instead a last column reporting the anticipation period with respect to the insurgency of the anomalies. As explained earlier, if there are at least three consecutive false positives before a Nagios-labeled failure, then we assume that the combined model is *predicting* the anomalous state, detecting that something in the computing node is not behaving properly *before the system administrators*. We calculate the anticipation period (reported in minutes in Tab. IV) by measuring the difference between the time stamp when the failure is labelled as such (via Nagios) and the time stamp of the first false positive in the series of consecutive ones leading to the anomaly.

The experimental results are really promising. Indeed, it is possible to anticipate the Nagios-annotated labels by a significant amount of time, roughly around 45 minutes when we compute the average over both January and May. The accuracy of the combined method is slightly inferior (or almost equal) compared to the semi-supervised one and definitely worse than the supervised approach. This is not an issue, as this is primarily due to the higher number of false positives detected before the label. This is a feature of the model we were aiming at: as stated before, we are not just interested in a classification method but we want to demonstrate the benefits in having an automated annotation tool and the capability to anticipate anomalies – and for this reason a lower detection accuracy

[20]Number of layers, number of neurons, learning rate, etc.

TABLE IV: Comparison of the experimental results obtained by combining the supervised and the semi-supervised approaches; average results over all nodes. The α value refers to the EMA computation (see Eq. 1). The anticipation is measured in minutes.

Month	α	TP	TN	FN	FP-Pre	FP-Post	FP-Rnd	FP	TPR	Prec.	FNR	F-score	Anticipation
Jan. 2020	0.3	19.70	5652.60	2.80	4.30	4.70	17.80	26.80	0.88	0.42	0.12	0.57	42 min
Jan. 2020	0.5	19.70	5664.10	2.80	2.70	4.70	7.10	14.50	0.88	0.58	0.12	0.69	33 min
Jan. 2020	0.7	19.70	5668.50	2.80	1.90	4.70	3.50	10.10	0.88	0.66	0.12	0.75	33 min
Jan. 2020	0.9	19.70	5669.50	2.80	1.50	4.70	2.90	9.10	0.88	0.68	0.12	0.77	33 min
May 2020	0.3	186.40	6901.00	8.30	10.60	9.70	63.10	83.40	0.96	0.69	0.04	0.80	58 min
May 2020	0.5	186.40	6923.60	8.30	10.20	9.60	40.80	60.60	0.96	0.75	0.04	0.84	57 min
May 2020	0.7	186.40	6930.10	8.30	10.20	9.60	34.40	54.20	0.96	0.77	0.04	0.86	57 min
May 2020	0.9	186.40	6932.20	8.30	10.00	9.60	32.50	52.10	0.96	0.78	0.04	0.86	57 min
Jan. & May	0.3	206.10	12553.60	11.10	14.90	14.40	80.90	110.20	0.95	0.65	0.05	0.77	50 min
Jan. & May	0.5	206.10	12587.70	11.10	12.90	14.30	47.90	75.10	0.95	0.73	0.05	0.83	45 min
Jan. & May	0.7	206.10	12598.60	11.10	12.10	14.30	37.90	64.30	0.95	0.76	0.05	0.85	45 min
Jan. & May	0.9	206.10	12601.70	11.10	11.50	14.30	35.40	61.20	0.95	0.77	0.05	0.85	45 min

is a reasonable price to pay.

VI. Conclusion

This work is an initial exploration towards using automatically annotated data and DL models for anomaly detection and prediction in HPC systems. We rely on three main elements: 1) a fine-grained holistic monitoring infrastructure (ExaMon in our case); 2) a SW tool which can produce labels distinguishing between normal and anomalous states (we employed Nagios for this purpose); 3) a DL model merging the strengths of semi-supervised and supervised approaches. To the best of our knowledge, we are the first to demonstrate how Nagios-annotated data can be used for this task, without requiring any change to the typical workflow of system administrators. The key result of the proposed approach is the possibility to anticipate the insurgence of faults with significant advance, on average between 40 and 50 minutes. This is a boon for predictive maintenance applications targeting exascale as corrective measures (e.g., proactive checkpoints, workload balancing or health checking routines) can be undertaken before reaching a critical state.

In future works we will continue to analyse the ever increasing data sets which are currently being collected by ExaMon on several supercomputers at CINECA, in particular the current tier-0 HPC system, Marconi100. Our objective is to replicate our findings on a different system exploiting a larger amount of data. We will investigate in more detail the cause of the anomalies, with the goal of better understanding their sources and possibly classify them in different categories (this will also provide a feedback on the preliminary root-cause analysis described in this paper). In this direction, we also plan to store and analyse historical log traces which can provide useful insights regarding supercomputing node failures.

Acknowledgment

This research was partly supported by the EU H2020-ICT-11-2018-2019 IoTwins project (g.a. 857191), the H2020-JTI-EuroHPC-2019-1 Regale project (g.a. 956560) and Emilia-Romagna POR-FESR 2014-2020 project "SUPER: SuperComputing Unifier Platform – Emilia-Romagna". We also thanks CINECA for the collaboration and access to their machines and Francesco Beneventi for maintaining ExaMon.

References

[1] S. Heldens, P. Hijma, B. V. Werkhoven, J. Maassen, A. S. Z. Belloum, and R. V. Van Nieuwpoort, "The landscape of exascale research: A data-driven literature analysis," *ACM Comput. Surv.*, vol. 53, no. 2, Mar. 2020. [Online]. Available: https://doi.org/10.1145/3372390

[2] J. Dongarra, "Report on the fujitsu fugaku system," 2020.

[3] "Top500list," 2020, https://www.top500.org/.

[4] L. A. Parnell, D. W. Demetriou, V. Kamath, and E. Y. Zhang, "Trends in high performance computing: Exascale systems and facilities beyond the first wave," in *2019 18th IEEE Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm)*, 2019, pp. 167–176.

[5] F. Shoji, "BWorld Robot Control Software," SC19 EEHPCWG Annual Meeting, 2019.

[6] C. Di Martino, W. Kramer, Z. Kalbarczyk, and R. Iyer, "Measuring and understanding extreme-scale application resilience: A field study of 5,000,000 hpc application runs," in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2015, pp. 25–36.

[7] M. S. Bouguerra, A. Gainaru, L. B. Gomez, F. Cappello, S. Matsuoka, and N. Maruyama, "Improving the computing efficiency of hpc systems using a combination of proactive and preventive checkpointing," in *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, 2013.

[8] A. Netti, Z. Kiziltan, O. Babaoglu, A. Sîrbu, A. Bartolini, and A. Borghesi, "A machine learning approach to online fault classification in hpc systems," *Future Generation Computer Systems*, 2019.

[9] O. Tuncer, E. Ates, and et al., "Diagnosing performance variations in hpc applications using machine learning," in *International Supercomputing Conference*. Springer, 2017, pp. 355–373.

[10] O. Tuncer, E. Ates, and Y. e. a. et Zhang, "Online diagnosis of performance variation in hpc systems using machine learning," *IEEE Transactions on Parallel and Distributed Systems*, 9 2018.

[11] A. Borghesi, A. Bartolini, and et al., "Anomaly detection using autoencoders in hpc systems," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019.

[12] A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, and L. Benini, "A semisupervised autoencoder-based approach for anomaly detection in high performance computing systems," *Engineering Applications of Artificial Intelligence*, vol. 85, pp. 634–644, 2019.

[13] M. Ott, W. Shin, and et al., "Global experiences with hpc operational data measurement, collection and analysis," in *2020 IEEE International Conference on Cluster Computing*, 2020.

[14] A. Netti, M. Muller, and et al., "Dcdb wintermute: Enabling online and holistic operational data analytics on hpc systems," in *Proc. of the 29th International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 101–112.

[15] A. Bartolini, F. Beneventi, and et al., "Paving the way toward energy-aware and automated datacentre," in *Proceedings of the 48th International Conference on Parallel Processing: Workshops*, 2019, pp. 1–8.

[16] W. Barth, *Nagios: System and network monitoring*. No Starch Press, 2008.

[17] G. Moschini, R. Houssou, J. Bovay, and S. Robert-Nicoud, "Anomaly and fraud detection in credit card transactions using the arima model," 2020.

[18] K. B. Lee, S. Cheon, and C. O. Kim, "A convolutional neural network for fault classification and diagnosis in semiconductor manufacturing processes," *IEEE Transactions on Semiconductor Manufacturing*, vol. 30, no. 2, pp. 135–142, 2017.

[19] T. Salman, D. Bhamare, A. Erbad, R. Jain, and M. Samaka, "Machine learning for anomaly detection and categorization in multi-cloud environments," 2018.

[20] M. Molan, "Pre-processing for Anomaly Detection on Linear Accelerator. CERN openlab online summer intern project presentations," Sep 2020.

[21] M. Gamell, K. Teranishi, and et al., "Modeling and simulating multiple failure masking enabled by local recovery for stencil-based applications at extreme scales," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 10, 2017.

[22] E. Meneses, X. Ni, and et al., "Using migratable objects to enhance fault tolerance schemes in supercomputers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 7, pp. 2061–2074, 2015.

[23] I. Boixaderas, D. Zivanovic, and et al., "Cost-aware prediction of uncorrected dram errors in the field," in *2020 SC20: International Conference for HPC, Networking, Storage and Analysis (SC)*. Los Alamitos, CA, USA: IEEE Comp. Soc., nov 2020.

[24] G. Iuhasz and D. Petcu, "Monitoring of exascale data processing," in *2019 IEEE International Conference on Advanced Scientific Computing (ICASC)*, 2019, pp. 1–5.

[25] X. Yang, Z. Wang, J. Xue, and Y. Zhou, "The reliability wall for exascale supercomputing," *IEEE Transactions on Computers*, vol. 61, no. 6, pp. 767–779, 2012.

[26] G. Pang, C. Shen, and et al., "Deep learning for anomaly detection: A review," 2020.

[27] G. Lemaître, F. Nogueira, and C. K. Aridas, "Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning," *Journal of Machine Learning Research*, vol. 18, no. 17, pp. 1–5, 2017.

[28] S. Bhatia, A. Jain, and B. Hooi, "Exgan: Adversarial generation of extreme samples," 2020.

[29] A. Netti, Z. Kiziltan, and et al., "Finj: A fault injection tool for hpc systems," in *European Conference on Parallel Processing*. Springer, 2018, pp. 800–812.

[30] A. Borghesi, A. Libri, and et al., "Online anomaly detection in hpc systems," in *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems*. IEEE, 2019.

[31] S. Ghiasvand and F. M. Ciorba, "Anomaly detection in high performance computers: A vicinity perspective," in *2019 18th International Symposium on Parallel and Distributed Computing (ISPDC)*. IEEE, 2019, pp. 112–120.

[32] G. Ostrouchov, D. Maxwell, and et al., "Gpu lifetimes on titan supercomputer: Survival analysis and reliability," in *2020 SC20: International Conference for HPC, Networking, Storage and Analysis (SC)*. Los Alamitos, CA, USA: IEEE Computer Society, nov 2020, pp. 568–581.

[33] S. Di and F. Cappello, "Adaptive impact-driven detection of silent data corruption for hpc applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 10, 2016.

[34] T. L. Groves, R. E. Grant, and et al., "Unraveling network-induced memory contention: Deeper insights with machine learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 8, pp. 1907–1922, 2018.

[35] F. Beneventi, A. Bartolini, C. Cavazzoni, and 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] "Kairosdb a fast scalable time series database," https://github.com/kairosdb/kairosdb, accessed: 2020-08-02.

[37] Apache, "Apache cassandra," https://http://cassandra.apache.org/, accessed: 2019-01-04.

[38] M. A. Jette, A. B. Yoo, and M. Grondona, "Slurm: Simple linux utility for resource management," in *In Lecture Notes in Computer Science: Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP) 2003*. Springer-Verlag, 2002.

[39] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," California Univ San Diego La Jolla Inst for Cognitive Science, Tech. Rep., 1985.

[40] CINECA, "Marconi," https://www.hpc.cineca.it/hardware/marconi, 2016, accessed: 2020-11-13.

[41] A. Netti, Z. Kiziltan, O. Babaoglu, A. Sîrbu, A. Bartolini, and A. Borghesi, "Online fault classification in hpc systems through machine learning," in *European Conference on Parallel Processing*. Springer, 2019, pp. 3–16.

[42] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

Andrea Borghesi is an Assistant Professor at the Department of Computer Science and Engineering (DISI) of the University of Bologna. His research focuses on optimization techniques and ML approaches for complex systems, especially in the area of HPC systems. He is also executive scientific representative for the HPC at the Interdepartments Center for AI at University of Bologna (ALMA-AI)

Martin Molan Martin Molan is a PhD student of data science and computation at University of Bologna. He has received BA in mathematics at University of Ljubljana and MA in ICT at JSI institute. As a student he has collaborated with CERN openlab, UCL center for AI, UNESCO International Research Center On Artificial Intelligence, and CINECA.

Michela Milano is full professor at DISI, University of Bologna and the director of the ALMA-AI institute. Her research activity concerns AI with focus on decision support and optimization systems covering both theoretical and practical aspects in application fields as energy, computing, and sustainability. She has edited two collections on hybrid optimization and she is author of more than 170 papers on peer reviewed international conferences and journals.

Andrea Bartolini is Assistant Professor in the Department of Electrical, Electronic and Information Engineering Guglielmo Marconi (DEI) at the University of Bologna. Before, he was Post-Doctoral researcher in the Integrated Systems Laboratory at ETH Zurich. He has published more than 120 papers in peer-reviewed international journals and conferences and several book chapters with focus on dynamic resource management - ranging from embedded to large scale HPC systems.