An Explainable Model for Fault Detection in HPC Systems*

Martin Molan[1][0000-0002-6805-2232], Andrea Borghesi[1][0000-0002-2298-2944], Francesco Beneventi[1], Massimiliano Guarrasi[2][0000-0002-3175-4628], and Andrea Bartolini[1][0000-0002-1148-2450]

University of Bologna, Italy[1]
CINECA, Italy[2]
{martin.molan2,andrea.borghesi3,francesco.beneventi,a.bartolini}@unibo.it
m.guarrasi@cincea.it

Abstract. Large supercomputers are composed of numerous components that risk to break down or behave in unwanted manners. Identifying broken components is a daunting task for system administrators. Hence an automated tool would be a boon for the systems resiliency. The wealth of data available in a supercomputer can be used for this task. In this work we propose an approach to take advantage of holistic data centre monitoring, system administrator node status labeling and an explainable model for fault detection in supercomputing nodes. The proposed model aims at classifying the different states of the computing nodes thanks to the labeled data describing the supercomputer behaviour, data which is typically collected by system administrators but not integrated in holistic monitoring infrastructure for data center automation. In comparison the other method, the one proposed here is robust and provide explainable predictions. The model has been trained and validated on data gathered from a tier-0 supercomputer in production.

Keywords: Machine Learning · High Performance Computing · Fault Detection

1 Introduction

High Performance Computing (HPC) systems are large machines composed by hundreds of thousands (up to millions) of smaller components (both software and hardware), all interacting in complex manners. A key challenge to be addressed by researchers in this area 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. The large scale of the problem motivates the development of an automated procedure for anomaly detection in current supercomputers, and this need will become even more pressing for future Exascale systems [25].

* Supported by University of Bologna and CINECA, Italy

In this situation, a great help comes from the fact that the performance and operative status of HPC systems are continuously monitored by different reporting and monitoring services, that gather data from software and hardware components [15]. Thanks to this data, it is possible to evaluate the health status of the system. These monitoring services are designed for and used by HPC system administrators who monitor the operation of the whole system, and who can manually disable certain parts to prevent serious damage to hardware components, or negative effects to system availability.

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 [2]. These services/software tools warn system administrators about critical conditions; system administrators can then verify the automatically generated alarms by manually inspecting the system status. Based on the result of the inspection, it is decided if it was a false alarm or if the compute node has to be "drained" from the production. Node downtime is recorded in logs and it is then used for post-mortem analysis. The current trend in data center information is systematic recording of system information data such as data coming from physical sensors' telemetry (temperature, power), micro-architectural events (IPC, cache misses), data coming from the computing resources and infrastructure data from cooling and power equipment (DCIM) [6, 19, 5]. This data is stored in the form of multivariate time series. However, in current HPC systems service-reporting tools are decoupled from physical monitoring infrastructure. *The first contribution of this paper is combining both the traditional (based on reporting services) and industry 4.0 (based on granular) data sources.* Concretely: we extend the Examon [6, 5] framework with Nagios data [3]. Our goal is to detect critical HPC node failures that are recorded by system administrators via Nagios as DOWN+DRAIN events (in the rest of the paper *system failure* and *label* is synonymous with DOWN+DRAIN event).

Using this data we build the second contribution of this work: *an explainable fault detection/classification model based on Machine Learning* (ML). The model will classify HPC node state depending on the monitored data. We call the model *TrueExplain* as it is based on *TrueSkill* model originally developed by Microsoft research [13]. This model benefits the administrators of the HPC system in two ways. First, it automates fault detection; this could shorten the elapsed time between recognition of system failure and reaction by administrators. Secondly, the proposed model provides *insights* into the dynamics of HPC system operation. The *insight generation* capability of the ML model can support the decision process that hinges on the model's predictions. Identified root cause of the suspected fault can be interpreted by a human user. Explainable ML models can also be more reliable as they can be validated by a domain expert. Despite explainable AI not being a novel concept, this is the first time it has been applied to resiliency in HPC systems and trained on real data from HPC production.

As a final contribution, it must be stressed out that the model was trained and validated using labeled data collected on a tier-0 supercomputer hosted by

the Italian Supercomputing Center (Cineca) [1]. We dealt with the real anomalies from a production machine unlike most approaches in the literature that deal with "synthetic" or artificially injected anomalies.

2 Related Works

Previous research works have made preliminary exploration towards the creation of model for fault detection in HPC systems. There have been approaches based on large quantity of data with scarce labels. These approaches belong to the *semi-supervised* field. For instance, Borghesi et al. [7–9] propose the usage of a particular type of Deep Neural Network (DNN) called autoencoder to learn the characteristic behaviour of a supercomputer in healthy state. These trained DNNs are then used to classify between normal and anomalous points in incoming data streams.

When the labels are available researchers have employed *supervised* ML-inspired algorithms to perform anomaly detection with high accuracy. Tuncer et al. [24] deal with the problem of diagnosing performance variations in HPC systems, using labeled data collected from a HPC simulator. The authors train different ML algorithms to classify the behaviour of the supercomputer using the gathered data. In a similar fashion, Netti et al. [18] propose a model based on Random Forest to classify different types of faults that can happen in a HPC node.

The previously described methods have been proven to be useful and have good accuracy. The important difference between existing work and the work presented in this paper is the use of data from real HPC production. Existing work such as [18] uses anomaly injection to *artificially* create faults that are then approximated with a machine learning model. As such there can be doubt about the generalization capabilities of such models; ML approach that is capable of recognising the dynamics and characteristics of fault injection 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 the dynamics of injected anomalies. Additionally, from a ML perspective, training on a real dataset brings additional challenges in the form of noise in the data. Instead, in this work, we deal with real, noisy data.

Additionally we aim at a fault detection model whose predictions could be explained. We have thus obtained a tool for data center automation that can be adopted by system administrator and facility owners more willingly, as it is possible to interpret its decision on a human-understandable way. The outputs of explainable models can be cross-referenced and checked by domain experts. Black box models can never truly be validated by domain experts - the very prominent (and in recent years popular with researchers) filed of research that deals with *fairness* and *bias* in artificial intelligence is largely the result of the inherent impossibility to examine the rules learned by complex black-box models [16]. It is shown that just trusting the black box models can lead to models that make ultimately wrong decisions [16].

Explaining ML models has been a rich source of research works. A baseline explainable approaches are feature importance explanation models like [26]. Another possible approach are inherently explainable models [11]. Among explainable models, Bayesian models are very promising. In particular a recent work by Microsoft Research [13] proposes a scalable Bayesian approach called *True Skill*. The basic idea is similar to ELO rating in chess [21] and is based on predicted skill of two opponents and their previous interactions. As opposed to ELO rating, the skill is a Gaussian variable (not a scalar value). The update of this variable depends both on it's mean and it's variance. The skill estimates (model priors) are updated so that the score of the winner is increased and score of the loser is decreased. The score update (increase for the winner and decrease for the looser) depends on the relative skills of the opponents. In general, this process could require a lot of previous data (not available for new players) and to overcome this limitation TrueSkill models player score as a Gaussian random variable instead of as a scalar value; the additional information obtained via the probability distribution – in particular the variance – allows for the more efficient update of players' scores.

The basic idea of TrueSkill can be adopted to other problems where interaction can be modeled as opposition between two agents. For instance, *TrueLearn* was developed by Bulathwela and et al. [10] to model the learning path of students consuming educational materials. The opposing agents are the learner and the material; the estimated parameters are material difficulty and learner's skill. Similarly, Molan et al. generalize TrueSkill [17] to assist blind students. In this case, the model learns student's accessibility preferences and uses those preferences to rank materials in terms of suitability. The idea of two opposing agents in Bayesian learning can also be abstracted for classification settings. This method was explored by Graepel et al. [12] to learn user preference with regards to different types of ads.

In this paper, we will extend the usage of TrueSkill to the context of fault classification in HPC systems, as described in the following section.

3 Methodology

The aim of this work is to construct an explainable ML model, trained on real data that I) is accurate and robust and II) provides justification for its decision. Robust models stay the same (and have the same predictions) when they are trained multiple times on the same dataset. We limit our focus on the binary classification, that is distinguishing the normal state of a HPC node from a faulty one. For this scope, the critical element is the identification of the most relevant attributes and of their relative contributions. To this end, we develop a Bayesian approach based on TrueSkill [13] aimed at estimating the relative importance of the input features. We name this approach *TrueExplain*. TrueExplain is compared to the standard for explainable classification models: decision trees. Decision trees serve as a baseline against which our novel ML apprach will be evaluated.

As a classifier, we opted to implement DTs as a baseline as they are simple to implement and tend to provide good accuracy [24]. Furthermore, in this work we are not only interested in the classification accuracy *per se*, but rather on the explainability of the model – and DTs are notoriously more "transparent" compared to model such as neural networks. Secondarily, we want to obtain a robust model, that is a model capable of making consistent predictions over multiple runs.

TrueExplain is an extension of *TrueSkill* that can be used for classification. It is a form of Bayesian method, that aims to increase the robustness of the overall approach and provide the explainability in the form of feature importance. The feature importance is modeled as a Gaussian variable with its mean and variance; the variance can be interpreted in terms of uncertainty about the model prediction. Additionally, the Bayesian approach True Explain is more robust and has consistent classification performance across different subsets of data – which is not necessarily the case for decision trees.

Our approach TrueExplain exploits the the Bayesian approach introduced by TrueSkill – it is an *opposition-based classifier*. Binary classification can be interpreted as an opposition of two agents as presented in table 1. Currently, TrueExplain only works for one-hot encoded discrete (nominal) variables; generalization to continuous variables is a topic of ongoing research. Hence, TrueExplain expects data composed by a set of examples (or rows), each one possessing a collection of binary features assuming only values 0 or 1 and a binary label; if the original data set contains non-categorical data, one-hot encoding needs to be applied.

TrueExplain is trained through iterating over rows of the data set – the model is updated after each row (each example in the training set). In order to understand an update after a single row, it is important to understand how to construct two opposing teams from a single line in a data set. The binary classification problem (an example can represent either a normal data point or a fault) is presented as a "game" between two opposing teams (the core scheme of TrueSkll). One team is represented by active (hot) features (features with value 1) and it is described by a Gaussian variable whose parameters have to be learned through the training process. The other team is a dummy one described by a (deformed) variable with zero mean and zero variance. The dummy team is the same size as a feature team. Both teams are presented in Table 1. Information about the label is interpreted as the *outcome* of the opposition; if the label is 1 the feature team won - otherwise the dummy team won.

Table 1: Representation of two opposing teams. G(0,0) designates a deformed Gaussian distribution. G(f) denotes a learned Gaussian for a feature f.

G(f2)	G(f3)	G(f5)	...	G(f(N-1))
		vs.		
G(0,0)	G(0,0)	G(0,0)	...	G(0,0)

The main advantage of using the scalable Bayesian approach as a classifier are interpretability and robustness. A Gaussian variable is associated to each feature and its parameters are learned during training. The mean value represents the relative feature importance and the variance represents the estimated confidence.

4 Case Study: Marconi HPC System

A very important aspect for the anomaly detection approach is the availability of large quantity of data that monitors and describes the state of a supercomputer. To test our approach we take advantage of a supercomputer with an integrated monitoring infrastructure able to handle large amounts of data coming from several different sources. We opted for a supercomputer hosted by Cineca, Italy, named Marconi [14] (peak performance 20PFlops), already endowed with a holistic monitoring infrastructure called Examon [6, 4]. We collect three kinds of data: i) physical data measured with sensors; ii) workload information obtained from the job dispatcher; iii) information about the state of the system and its services collected by Nagios [3], a tool to provide alerts for system administrators. In this work we propose to use the information about the system services provided by Nagios to characterize Marconi's nodes. In Nagios the labels have been manually annotated by system administrators by reporting the nodes which experienced a failure in the considered time-frame. Nagios information consists of a set of categorical values for each alert probe set up during the configuration phase; this data type is well-suited for the one-hot encoding required by TrueExplain (see Sec. 3). To the best of the authors knowledge this is the first work proposing to use Nagios data integrated with a holistic monitoring infrastructure.

4.1 Dataset

The dataset consists of data collected from Marconi over a period of over 4 months[1]. The size of the collected data is 1GB. The data set is composed by combining high level status reporting information and corresponding system availability information – both recorded by Nagios reporting system. The label (target) consists of two values: 0 describing normal operation and 1 describing fault (DOWN+DRAIN event as recorded by Nagios). The label is provided by system administrators. On average we have recorded 1.5 faults per node (in the time period) with a maximum of 10 and minimum of 0 faults. The data sampling frequency is 15 minutes, as determined by Nagios monitoring functionalities. 15 minutes is the maximum possible sampling frequency. The data about a single computing node's availability (manual shutting down of nodes – our label), is referenced to subsystem availability reporting intervals – each 15 minute interval is annotated by information about node availability at that time interval.

All the data, discussed in this paper, was collected on the A1 partition of the Marconi system that consists of 1521 nodes with 36 cores and 128GB of

[1] April-July 2019

RAM. All nodes were manufactured by the same supplier. All nodes also report the same data back to the monitoring system. Since all the nodes are part of the same partition and were in full production, they were similarly frequently used. We have specifically chosen the observation period where all nodes were in production. We have no information about the job status on the nodes; this could be a useful additional information but it is the subject of further work.

4.2 Raw data pre-processing

The data is stored as system logs, one log for each service monitored by Nagios (including node availability – our label), and divided in multiple files depending on nodes and time stamp. This raw data format (shown in Table 2) is not suitable for training ML models. It has been reshaped by a pre-processing phase which included merging data from different Nagios services (including label data).

Table 2: Raw dataset as recorded in system logs.

Time stamp	Node	Reporting service	State
1256953732	r033c01s04	CPU temp	Warning (1)

To have a suitable training set for TrueExplain, raw data is transformed to an attribute-value description, that is each instance is described by values of selected attributes (and a target variable) [22]. The transformation produces the data in the new format summarized in Table 3.

Table 3: Section of attribute-value representation of data.

Node	Time Stamp	CPU temp	...	Downtime/Target
r033c01s04	1256953732	Warning (1)	...	Available (0)

For each time stamp and for each node, information for all reporting services are recorded alongside the log (downtime label) information. The downtime label also serves as a target class (label) while building the fault detection model. The data transformation is performed via parallel jobs executed on another supercomputer hosted at CINECA, namely Galileo HPC system. Galileo is also used for training and testing TrueExplain; Galileo is composed by 1022 nodes equipped with 2 x 18-cores Intel Xeon E5-2697 v4 (Broadwell) at 2.30 GHz. Overall, the data transformation was run on 60 nodes and it took a few minutes less than 3 hours.

4.3 Data description

The data set consists of 31 attributes and a target nominal attribute *Fault*. Attributes correspond to available monitoring services; Fault is a binary attribute

describing whether the system is at Fault at a given timestamp. Fault events are identified through the logs annotated by system administrators under normal Marconi usage – *we did not require any additional annotation operation besides those performed for typical system maintenance.* As each node will have its own fault detection model (trained and tested on its own data), the data is grouped accordingly. Two of the 31 attributes are then not informative for TrueExplain – timestamp and node [2] – and are discarded.

The input features/attributes are listed in Table 4. They represent the statuses of various subsystems of the supercomputer nodes. All 29 attributes are categorical attributes with three possible values: 0 denotes normal operational condition, 3 denotes a warning situation and 2 denotes a potentially serious (abnormal) condition[3]. These values come from configuration of Nagios on Marconi.

Table 4: Attributes used in construction of a model. The attributes are in the format - component::service::status or component::status.

alive::ping	backup::local::status	cluster::status::availability
cluster::status::criticality	cluster::status::internal	dev::raid::status
dev::swc::confcheckself	filesys::local::avail	filesys::local::mount
filesys::shared::mount	memory::phys::total	ssh::daemon
sys::ldap_srv::status	batchs::JobsH	filesys::eurofusion::mount
sys::gpfs::status	dev::ipmi::events	dev::swc::bntfru
dev::swc::bnthealth	dev::swc::bnttemp	dev::swc::confcheck
batchs::client::state	batchs::client	net::opa::pciwidth
net::opa	sys::orphaned_cgroups::count	core::total
sys::cpus::freq	batchs::client::serverrespond	

5 Experimental Results

In this work we focus on obtaining a model that is explainable and robust (and not necessarily the most accurate). We have trained and tested different standard supervised classification methods from the literature, namely: Decision Trees (DT), Linear Support Vector Machine (L-SVM), Nearest Neighbors (NN), Radial Basis Function SVM (RBF-SVM), and Random Forest (RF)[4]. All standard ML techniques are implemented in Python 3 (Python 3.6) using Scikit-learn [20]; TrueExplain is implemented in Python 3 as an extension of TrueSkill. The performance of different classification algorithms is evaluated on the pre-processed data set. All algorithms receive the same data that is processed in the same

[2] Time stamps uniquely identifies rows/examples

[3] Value 1 is unused

[4] Hyperparameters: DT max dept equal to none, splitting heuristic Gini impurity, min samples leaf equal to 1; L-SVM loss function squared hinge, regularization $l2$; NN number of neighbours equal to 5, uniform weights, euclidean metric; RBF-SVM has RBF kernel, regularization parameter equal to 1, RF number of estimators equal to 10, base estimator parameters same as DT

manner.F-score for the algorithms was calculated on a 20% test set. Each algorithm was trained and tested on 57 different nodes of Marconi. The average accuracy results (measured as F-Score) over all nodes are presented in Table 5.

Table 5: Average accuracy of classification methods across all nodes.

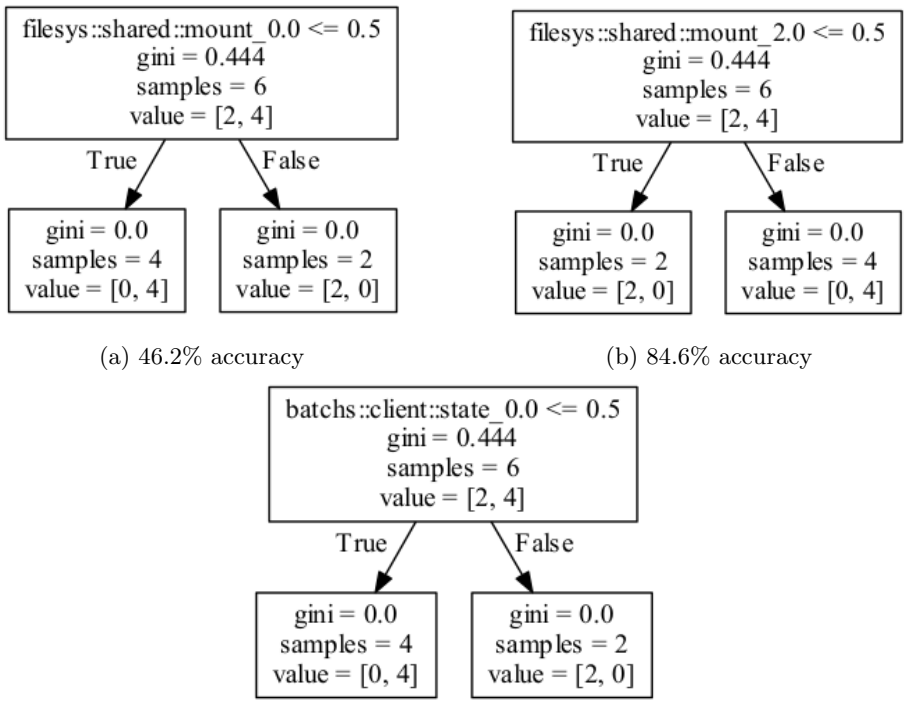
Decision Trees (DT)	0.97
Linear SVM	0.51
Nearest Neighbors	0.83
RBF SVM	0.66
Random Forest (RF)	0.91
TrueLearn	0.77

These experimental results reveal that TrueLearn is not the most accurate method. Its accuracy is on par with other methods, being outperformed only by DT-based classifiers. As mentioned earlier, in this work we do not aim at finding the most accurate ML model for fault detection, but we rather focus on obtaining a model that is explainable and *robust*. In the following section we discuss the robustness results; in particular we compare the robustness of TrueLearn with the robustness of the best explainable method, that is decision tree.

We do not measure the robustness since we are comparing a method that gives the same result in *every* run against the method (decision trees) that gives multiple different results. If we had multiple methods that produce different models between different runs, we could measure robustness of those methods by similarity of the models (e.g. similarity of decision tree graphs). In our case, robustness is the inherent characteristic of the design of the TrueLearn.

5.1 Fault Detection Robustness

Standard implementations of decision tree classifier [23] use random choice for splitting feature if more than one feature carry the same information about target label. This is the reason why decision trees, as well as their accuracy, can *change between different runs on the same dataset*. Results of 100 runs on the same dataset, validated by chronological 80/20 split are presented in a table 6. This is in stark contrast with TrueLearn which provides the *exact same accuracy level on all runs*; this is a significant boost in terms of model robustness and reliability. As seen in the table 6 the biggest difference between min, max and average accuracy is on the node $r104c14s02$. This node is the same as other nodes in terms of configuration but it serves as a good example of instability of decision trees. There are three possible induced decision trees on that dataset; the induced decision trees are presented on Figures 1a, 1b and 1c. All three possible decision tree graphs are equally likely and each one occurs in $\frac{1}{3}$ of the cases.



(a) 46.2% accuracy

(b) 84.6% accuracy

(c) 100% accuracy

Fig. 1: Different decision trees on node r104c14s02. Left node predicts class 1 (fault) and right one class 0 (no fault). Reported is the accuracy on the test set.

5.2 Visualization of learned parameters for Bayesian classifier TrueExplain

On the same node - node r104c14s02 - where decision trees experienced most fluctuation in terms of performance, TrueExplain classifier is also trained. Performance of TrueExplain classifier *is the same on all runs*. Distributions can also be visualized and plotted as Gaussians as presented in Figure 2.

6 Discussion

In general explainable classification models perform well compared to black box models. This is probably mostly the result of very informative features; reporting services (NAGIOS) already transform low level diagnostics data into higher level features. As such this dataset (beyond the frame of this work) could serve as a nice practical dataset for explainable classification models. Comparing the two explainable models (decision trees and Bayesian classifier TrueExplain) we see that decision trees outperform Bayesian approach in both the average and the

Table 6: Average, min and max accuracy in 100 runs of decision tree algorithm on the same dataset - results on selected nodes with average calculated on all nodes.

Node	average accuracy	max accuracy	min accuracy
r070c13s04	1.00	1.00	1.00
r075c13s02	0.85	1.00	0.53
r078c14s01	0.72	0.72	0.72
r090c14s02	1.00	1.00	0.99
r093c15s03	0.99	0.99	0.98
r094c04s03	1.00	1.00	1.00
r098c05s02	0.86	1.00	0.77
r103c11s02	0.86	0.86	0.86
r104c12s02	0.81	1.00	0.46
r104c14s02	0.83	1.00	0.46
r112c02s02	1.00	1.00	0.99
r145c10s04	0.99	1.00	0.98
r162c15s01	0.87	1.00	0.81
r169c11s04	0.70	0.97	0.65
r170c13s04	0.99	1.00	0.96
Average	0.974	0.992	0.951

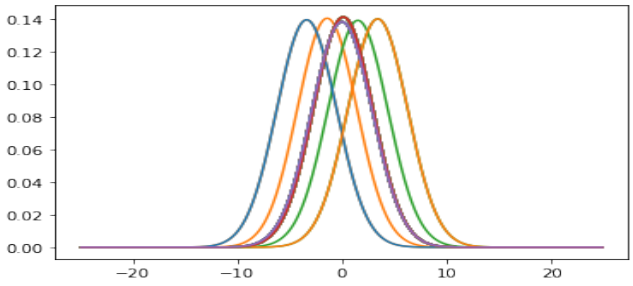


Fig. 2: Plotted weight distributions form TrueExplain, learned from node r104c14s02. The performance of TrueExplain is constant across different runs.

best case. It is the worst case scenario for Decision trees that is worrying - on this problem one has no guarantee that decision trees will be able to learn and produce usable predictions.

Exploring the most problematic node for decision trees - node r104c14s02 - illustrates the nature of the problem with decision trees. After training decision tree classifier on train dataset (80 percent of the data) it is apparent that the algorithm recognizes multiple features as equally informative on the *train set*. Thus the algorithm randomly chooses between three possible models (Figures 1a, 1b and 1c). Only model 1c however performs equally well on the test set. Diving into the nature of the underlying problem it can be assumed that different reporting services report errors simultaneously. Only one error is however the real

cause of the fault that can be generalized to the test set. Decision trees however have no guarantee of recognizing the real culprit. Described problem is especially characteristic for small dataset (per node models) and can be addressed by accumulating more data.

Bayesian model TrueExplain recognizes and increases the weights of all possible causes of a system failure. This system (on such a small dataset) protects against false negative errors that decision trees can make. The main advantage of the Bayesian approach is that it can recognize multiple factors as equally important in predicting a malfunction. For each attribute, TrueExplain learns two parameters μ (expected value) and σ. μ denotes expected contribution to class 1 (positive μ) or class 0 (negative μ). Additionally Bayesian learner TrueExplain also communicates prediction certainty. Variance (curve width) carries information about parameter certainty – as the certainty increases the width of the curve decreases (the Gaussian curve slowly tends towards a constant).

We are using the one-hot encoding for features; each feature is split into three sub-features according to it's critical state (0, 2, 3). Let us focus on a node where decision trees achieve poor performance: *r104c14s02*. Investigating the outputs of the TrueExplain for node r104c14s02, it can be seen that the model identifies as the most influential features (highest negative expected value - μ) contributing to class 0: *filesys::shared::mount*, *sys::gpfs::status*, *batchs::client::state*, all with critical state 0. Interestingly the same three factors, when they have critical state 2 (fault), are the most relevant to class 1. This means that, for this specific node, these three factors really determine if the node is available or not. In other words, these three factors explain the availability of examined node. Identification of pairs of important attributes (similar importance of the same feature with critical state 0 and critical state 2) is also in line with domain expectations for the problem. If a fault of a subsystem means the downtime of the node, than the normal operation of the subsystem should indicate availability of the node.

7 Conclusions

In this paper we tackle the problem of anomaly recognition in HPC systems. We propose a novel approach to characterize the behaviour of a supercomputing node, through the combination of I) measurements from hardware/software sensors and II) labels(system availability) provided by system administrators. Using this labeled data, we created and trained *TrueExplain*, a Bayesian classifier for distinguishing normal and anomalous states in supercomputing nodes. We have empirically shown that TrueExplain is I) robust, II) its decision offers a possibility for interpretation, and III) it works on smaller training set. The last point is very beneficial in terms of implementation on HPC systems, as it means that explainable and efficient models can be constructed for individual or multiple nodes relatively early on in the operational life of a new HPC system. Based on the work presented here, informative models for each individual node can be constructed in just a few months of operation. Another additional benefit of the

proposed Bayesian model is that it is inherently an online training algorithm – which means it can be trained in real time.

TrueExplain also can greatly help HPC system administrators as it accurately detects undesired states, thus improving the supercomputer resilience. This result is boosted by empirically proven robustness of the presented approach, as it is capable of consistently providing the same classification results over different nodes and hundreds of runs. Furthermore, the fact that our model is relatively simple to explain means that predictions can greatly help HPC operators in identifying the malfunctioning components and fault root causes.

8 Code access

Code for a classifier, implemented in this work is accessible at:

https://github.com/MolanM/TrueExplain

9 Acknowledgements

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 thank CINECA for the collaboration and access to their machines.

References

1. Cineca inter-university consortium web site, http://www.cineca.it//en, accessed: 2018-06-29
2. Sensu go: Sensu go 5.20, https://docs.sensu.io/sensu-go/latest/
3. Barth, W.: Nagios: System and network monitoring. No Starch Press (2008)
4. Bartolini, A., Borghesi, A., et al.: The D.A.V.I.D.E. big-data-powered fine-grain power and performance monitoring support. In: Proceedings of the 15th ACM International Conference on Computing Frontiers, Ischia, Italy, 2018 (2018)
5. Bartolini, A., Beneventi, F., Borghesi, A., Cesarini, D., Libri, A., Benini, L., Cavazzoni, C.: Paving the way toward energy-aware and automated datacentre. In: Proceedings of the 48th International Conference on Parallel Processing: Workshops. ICPP 2019, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3339186.3339215, https://doi.org/10.1145/3339186.3339215
6. Beneventi, F., Bartolini, A., et al.: Continuous learning of hpc infrastructure models using big data analytics and in-memory processing tools. In: Proceedings of the Conference on Design, Automation & Test in Europe. pp. 1038–1043. European Design and Automation Association (2017)
7. Borghesi, A., Bartolini, A., et al.: Anomaly detection using autoencoders in hpc systems. In: Proceedings of the AAAI Conference on Artificial Intelligence (2019)

8. Borghesi, A., Bartolini, A., et al.: A semisupervised autoencoder-based approach for anomaly detection in high performance computing systems. Engineering Applications of Artificial Intelligence **85**, 634–644 (2019)
9. Borghesi, A., Libri, A., et al.: Online anomaly detection in hpc systems. In: 2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS). pp. 229–233. IEEE (2019)
10. Bulathwela, S., Perez-Ortiz, M., et al: Truelearn: A family of bayesian algorithms to match lifelong learners to open educational resources. In: Proceedings of the AAAI Conference on Artificial Intelligence (2020)
11. Burkart, N., Huber, M.F.: A survey on the explainability of supervised machine learning. CoRR **abs/2011.07876** (2020), https://arxiv.org/abs/2011.07876
12. Graepel, T., Candela, J., et al.: Web-scale bayesian click-through rate prediction for sponsored search advertising in microsoft's bing search engine. Omnipress (2010)
13. Herbrich, R., Minka, T., Graepel, T.: Trueskill™: a bayesian skill rating system. In: Advances in neural information processing systems. pp. 569–576 (2007)
14. Iannone, F., Bracco, G., et al.: Marconi-fusion: The new high performance computing facility for european nuclear fusion modelling. Fusion Engineering and Design **129**, 354–358 (2018)
15. Massie, M.: Monitoring with Ganglia. O'Reilly Media, Sebastopol, CA (2012)
16. Mehrabi, N., Morstatter, F., Saxena, N., Lerman, K., Galstyan, A.: A survey on bias and fairness in machine learning. arXiv preprint arXiv:1908.09635 (2019)
17. Molan, M., Bulathwela, S., Orlic, D.: Accessibility recommendation system. In: Proceedings of the OER20: Open Education Conference (2020)
18. Netti, A., Kiziltan, Z., et al.: A machine learning approach to online fault classification in hpc systems. Future Generation Computer Systems (2019)
19. Netti, A., Mueller, M., Guillen, C., Ott, M., Tafani, D., Ozer, G., Schulz, M.: Dcdb wintermute: Enabling online and holistic operational data analytics on hpc systems (10 2019)
20. Pedregosa, F., Varoquaux, G., Gramfort, A., et Al.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
21. Pelánek, R.: Applications of the elo rating system in adaptive educational systems. Computers & Education **98**, 169 – 179 (2016)
22. Sammut, C., Webb, G.I. (eds.): Attribute-Value Learning. Springer US (2010)
23. Sharma, H., Kumar, S.: A survey on decision tree algorithms of classification in data mining. International Journal of Science and Research (IJSR) **5**(4) (2016)
24. Tuncer, O., Ates, E., et al.: Diagnosing performance variations in hpc applications using ml. In: International Supercomputing Conference. Springer (2017)
25. Yang, X., Wang, Z., Xue, J., Zhou, Y.: The reliability wall for exascale supercomputing. IEEE Transactions on Computers **61**(6), 767–779 (2012)
26. Zamuda, A., Zarges, C., Stiglic, G., Hrovat, G.: Stability selection using a genetic algorithm and logistic linear regression on healthcare records. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion. p. 143–144. GECCO '17, Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3067695.3076077, https://doi.org/10.1145/3067695.3076077