E2EWatch: An End-to-end Anomaly Diagnosis Framework for Production HPC Systems

Burak Aksar[1][0000−0003−3627−7311], Benjamin Schwaller[2], Omar Aaziz[2], Vitus J. Leung[2], Jim Brandt[2], Manuel Egele[1], and Ayse K. Coskun[1]

[1] Boston University, Boston MA 02215, USA {baksar,megele,acoskun}@bu.edu
[2] Sandia National Laboratories, Albuquerque NM 87123, USA
{bschwal,oaaziz,vjleung,brandt}@sandia.gov

Abstract. In today's High-Performance Computing (HPC) systems, application performance variations are among the most vital challenges as they adversely affect system efficiency, application performance, and cost. System administrators need to identify the anomalies that are responsible for performance variation and take mitigating actions. One can perform manual root-cause analysis on telemetry data collected by HPC monitoring infrastructures to analyze performance variations. However, manual analysis methods are time-intensive and limited in impact due to the increasing complexity of HPC systems and terabyte/day-sized telemetry data. State-of-the-art approaches use machine learning-based methods to diagnose performance anomalies automatically. This paper deploys an end-to-end machine learning framework that diagnoses performance anomalies on compute nodes on a 1488-node production HPC system. We demonstrate job and node-level anomaly diagnosis results with the Grafana frontend interface at runtime. Furthermore, we discuss challenges and design decisions for the deployment.

Keywords: HPC · Anomaly Diagnosis · Machine Learning · Telemetry.

1 Introduction

High-Performance Computing (HPC) systems offer invaluable computing resources for a range of scientific and engineering applications, such as national security, scientific discovery, and economic research. Following the massive growth in data and computing power, system infrastructure has grown more complex, and effective management of HPC systems has become more challenging. Many researchers report *anomalies* that cause performance variations due to network contention [9], hardware problems [23], memory-related problems (e.g., memory leak) [2], shared resource contention (e.g., reduced I/O bandwidth) [8, 17], or CPU-related problems (e.g., CPU throttling, orphan processes) [13]. Performance anomalies do not necessarily terminate the execution, but often increase job execution times by greater than 100% [22, 32, 37].

System administrators continuously collect and analyze system telemetry data with rule-based heuristics (e.g., [3, 13]) to determine the causes behind performance variations. Due to the highly complex infrastructure and massive volumes of telemetry data (e.g., billions of data points per day), rule-based methods

are incapable of effective management and analysis. Thus, researchers use machine learning (ML)-based tools more often to detect and diagnose performance variations automatically [4, 11, 15].

In this paper, we propose *E2EWatch*, an end-to-end anomaly diagnosis framework for production HPC systems. *E2EWatch* detects and diagnoses previously seen performance anomalies in compute nodes at runtime based on a recently proposed ML-based approach [33]. We design an end-to-end architecture for deployment and deploy this framework on a 1488-node HPC production system to display job and node level analysis results with an easy-to-interpret user interface. Our specific contributions are as follows:

- Deployment of a state-of-the-art anomaly diagnosis framework on a 1488-node production HPC system;
- Visualization and analysis of job and node-level anomaly diagnosis results on-the-fly;
- Demonstration of the effectiveness of our framework under a variety of experimental scenarios and discussion of deployment challenges and techniques.[3]

The rest of the paper is organized as follows. Section 2 provides a brief overview of related work; Sec. 3 describes the methodology in detail; Sec. 4 describes experimental scenarios; Sec. 5 presents our results, and we conclude in Sec. 6.

2 Related Work

Performance variation has been an important research topic for large-scale computing systems. Especially as we move towards the exascale computing era, it will remain a substantial challenge. This section briefly reviews the latest anomaly detection and diagnosis research in three categories: rule-based statistical methods, ML-based methods, and deployment.

Rule-based statistical methods: These methods are widely used in large-scale production systems since they are generally easier to design and deploy in practice compared to more sophisticated ML-based methods. Some example methods use manually selected threshold values for important system metrics [3, 18]. Some researchers investigate the statistical correlation between features and performance issues instead of solely assigning thresholds. Brandt et al. track systems and components' operational behaviors over time and analyze their correlations with various causes, such as aging components [13]. Agelastos et al. leverage system-wide resource utilization data and investigate specific metrics to detect I/O congestion and out-of-memory cases [2]. Even though many rule-based statistical methods are easy to implement for the administration side, their efficacy is highly dependent on system properties (e.g., operating system, underlying hardware configurations).

ML-based methods for performance analytics: Some researchers focus on predicting node or application-level failures using ML models trained on system monitoring data and logs [16, 20, 35]. Ates et al. use ML models to detect

[3] Our implementation is available at: https://github.com/peaclab/E2EWatch

applications running on supercomputers by leveraging applications' resource utilization characteristics [5]. Anomaly detection (for a broader range of events than failures) is widely popular in the HPC and cloud domains [26, 36, 10]. However, most anomaly detection focuses on detecting anomalies instead of providing information on the anomaly type. Several ML approaches have been proposed to detect anomalous behavior in applications and compute nodes using historical *normal* data [4, 11, 15, 21, 31]. Tuncer et al. leverage historical telemetry data to diagnose previously observed performance anomalies on compute nodes during an application run, but do not demonstrate a runtime deployment [33]. In addition to node-level anomaly detection and diagnosis, Xie et al. train a one-class support vector machine on vector embeddings to detect anomalous function executions using call stack trees [34]. In another work, Denis and Vadim use a Long Short Term Memory (LSTM) network to detect abnormal and suspicious behavior during an application run [31].

Deployment: Operational Data Analytics (ODA) solutions provide runtime system insights for users and system administrators and complement monitoring frameworks [27, 30, 12]. Some important application areas of ODA are application fingerprinting, scheduling and allocation, performance variation detection. Netti et al. demonstrate the use of several ML models to forecast compute node power and identify outliers and anomalous behavior using power, temperature, and CPU metrics on an HPC cluster at runtime [27]. Borghesi et al. propose an autoencoder-based semi-supervised approach to detect anomalous behaviors in compute nodes and deploy them to their 45-node HPC system [11].

In production systems, the size of a system can substantially affect the deployment because a production HPC system might have thousands (e.g., Sierra, Astra) to tens of thousands (e.g., Cori, Blue Waters) compute nodes. For example, Borghesi et al. train node-specific models that use monitoring data from a specific node and a node-agnostic model that uses monitoring data from all compute nodes during training [11]. Using node-specific models could be feasible for small computing clusters; however, it infers high training and maintenance costs, e.g., selecting a new detection threshold for each model is time-consuming. The abovementioned approaches (e.g., [11, 27]) only collect data when the system behaves in the normal state, which requires constant system assessment by a system administrator. A manual assessment approach may not be feasible considering the complexity of HPC systems. Another aspect is evaluating models' performance against scenarios where there are unknown applications and unknown application inputs in order to guarantee they perform as intended. However, neither of the methods covers real-world deployment scenarios. Even though Netti et al. and Borghesi et al. have online deployment components, they solely focus on detecting anomalies rather than classifying their type [11, 27]. To the best of our knowledge, none of the prior methods provide an end-to-end anomaly diagnosis framework running on a large-scale production HPC system.

3 Methodology

The main goal of *E2EWatch* is to diagnose the root cause of previously observed performance anomalies in compute nodes during application runs. We provide diagnosis results in a dashboard that enables users or system administrators to track their applications' status and interfere when necessary. Figure 1 shows an overview of *E2EWatch*. In this paper, we focus on anomalies that cause performance variability (e.g., CPU contention and memory problems) in different sub-systems instead of faults that terminate the execution of a program prematurely. Our framework can diagnose the anomaly type and provide easy-to-interpret results to users while an application is still running.

3.1 *E2EWatch* Overview

E2EWatch is an end-to-end anomaly diagnosis framework similar to Tuncer et al.'s framework, and we deploy the framework on a production HPC system [33]. *E2EWatch* has a user interface and works with labeled data that system administrators or automated methods can generate. In this work, we collect system telemetry data from compute nodes while running applications with and without synthetic anomalies that produce well-known performance variations (e.g., CPU contention, memory leakage). Specifically, we collect resource usage (e.g., free memory, CPU utilization) and performance counter telemetry data (e.g., CPU interrupt counts, flits) across a set of applications. After the data collection phase, we apply statistical preprocessing techniques (e.g., feature selection) to raw time series data to extract useful information and then train supervised ML models. We compare the performance of a set of ML models in the test data and deploy the best model to the *monitoring server*. At runtime, a user queries a specific job-id assigned by the monitoring server. We provide a summary across all compute nodes, and drill-down analysis for each node, used by the application. In the upcoming sections, we explain each phase in detail.

3.2 Offline Data Collection

The goal of offline data collection is to collect high-fidelity monitoring data to train ML models. We use Lightweight Distributed Metric Service (LDMS) to collect telemetry data across different subsystems [1]. We run controlled experiments with synthetic anomalies from the HPC Performance Anomaly Suite (HPAS) with three real and three proxy applications to mimic performance anomalies [6]. We perform data cleaning and interpolation for missing hardware metrics and increasing counter values. We provide the details of the anomalies, applications, and data preprocessing in Sec. 4.

3.3 Offline Data Preparation

After the offline data collection, we divide raw time series into multiple equal-length overlapping windows with 15-seconds skip intervals (e.g., [0-45], [15-60]). While the windowing operation substantially increases the amount of training

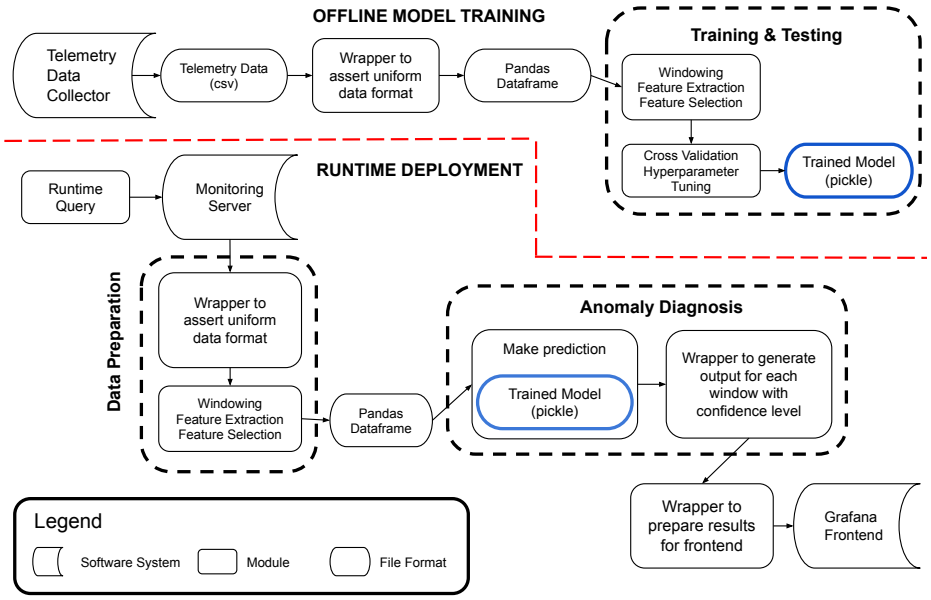


Fig. 1. The high-level architecture of *E2EWatch*. The top flow contains the offline training phase, and the bottom flow contains the online anomaly diagnosis phase. We train ML models with known normal and anomalous application telemetry data and find the best parameters with hyperparameter tuning for each model. We deploy the best-performing model to the monitoring server. A user sends a query with the specific application ID at runtime, and our framework displays node-by-node diagnosis results in the Grafana frontend interface.

data, it also enables us to provide results without waiting for an application to run to completion. Then, we calculate the following statistical features of each window: "minimum and maximum; 5^{th}, 25^{th}, 50^{th}, 75^{th}, 95^{th} percentile values; mean, variance, skewness and kurtosis" [33]. Each 2D window transforms into a 1D vector after the feature extraction stage. This approach brings substantial savings in computational power and memory while preserving the main characteristics of time series.

After we extract statistical features, we adopt the feature selection process proposed by Tuncer et al. [33]. We calculate the cumulative distribution functions (CDF) of each feature when running the application with and without an anomaly. We use *the Kolmogorov-Smirnov* (KS) test to compare each feature's CDF [24]. CDFs of normal and anomalous metric values show a high statistical difference if the anomaly substantially affects metric values. We repeat the abovementioned procedure and concatenate the selected features for each application and anomaly pair.

3.4 Offline Training and Testing

The goal of this stage is to find the best-performing model before deploying the model to the monitoring server. After the offline data preparation phase, we perform hyperparameter tuning for each ML model and test their performance using multiple cross-validation folds. We label each equal-sized window for our anomaly diagnosis task according to the node's label it belongs to. For example, if we run an anomaly on a compute node, we label it according to the type of the anomaly; otherwise, it is labeled *normal*.

3.5 Deployment and Runtime Diagnosis

We use the model trained during the offline training phase for runtime diagnosis. We store the trained model on the monitoring server of our target system (see Sec. 4 for details) along with other back-end components such as data storage and visualization. At runtime, a user sends a query to the monitoring server with the desired application ID assigned by the HPC system scheduler, and our runtime analysis module presents results in the Grafana frontend. For the job-level breakdown, we calculate how many anomalous windows are diagnosed for each anomaly across the compute nodes used by the application (e.g., 75% of windows have "cachecopy" anomaly). In Fig. 2, we demonstrate a job-level diagnosis summary for one example application run. The user can see all anomalies diagnosed over time along with the classifier's prediction confidence for the selected application ID. We explain the calculation of the prediction confidences in Sec. 5. Furthermore, it is possible to perform drill-down analysis for the selected compute nodes.

4 Experimental Methodology

We detail the target system and the monitoring framework in the first section. Next, we describe the synthetic anomalies we use to create performance variation. In the last section, we explain the implementation details of *E2EWatch*.

4.1 Target System and Monitoring Framework

To demonstrate the efficacy of *E2EWatch*, we conduct experiments and deploy the framework on Eclipse, a production HPC system with 1488 compute nodes located at Sandia National Laboratories (SNL). Each node has 128GB memory and two sockets, where each socket has 18 E5-2695 v4 CPU cores with 2-way hyperthreading [29]. System administrators use LDMS to monitor the system health of Eclipse, and LDMS is actively running on all compute nodes. LDMS can collect thousands of different resource usage metrics and performance counters from compute nodes at sub-second granularity. From LDMS, we use 160 system metrics sampled at 1Hz while running six applications with and without synthetic anomalies. The applications used are listed in Table 1. SNL has a separate monitoring server, referred to here as HPCMON, for data storage, analysis, and visualization. HPCMON is a four-node cluster with 48 Intel Xeon

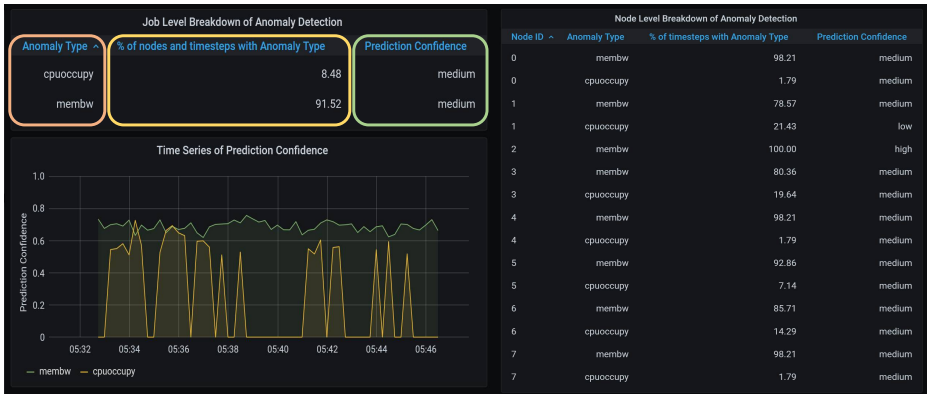


Fig. 2. An example SWFFT application run with the *membw* anomaly. We provide diagnosed anomaly types (orange box) and their percentage across all windows (yellow box) as well as model's prediction confidences (green box) in the job-level breakdown. The time series plot shows the average confidence level of each anomaly across all the compute nodes used by the application. We also provide the node-level breakdown composed of the same analyses we provide in the job-level breakdown.

Gold 6240 CPUs, 750 GB of memory, and 28 TB of NVMe raided storage. Eclipse telemetry data is sent to HPCMON and is queryable through a Grafana frontend interface. A user specifies the time range and application ID of interest, and the corresponding telemetry data is sent through our ML models at query time. The model output is then summarized and formatted for Grafana visualization [30].

4.2 Synthetic Anomalies

We use open-source synthetic anomalies from High-Performance Anomaly Suite (HPAS) [6]. HPAS has 8 performance anomalies that create contention across different subsystems such as memory, network, and I/O. We use the following anomalies: *memleak*, which mimics memory leakage by allocating an array of characters of a given size without storing the addresses; *membw* mimics memory bandwidth contention, which prevents data from being loaded into the cache; *cpuoccupy* mimics excessive CPU utilization; *cachecopy* mimics cache contention by allocating two-arrays and swapping their contents repeatedly for a specific

Table 1. We run 3 real and 3 proxy applications during offline data collection phase.

Benchmark	Application	Description
ECP Proxy Suite	ExaMiniMD	Molecular dynamics
	SWFFT	3D Fast Fourier Transform
	sw4lite	Numerical kernel optimizations
Real Applications	LAMMPS	Molecular dynamics
	HACC	Cosmological simulation
	sw4	Seismic modeling

cache level, e.g., L3 cache. We have 3 different input configurations for each application, which corresponds to 4, 8, and 16 node application runs, respectively. On each node, we allocate one core for LDMS, one core for an anomaly, and 32 cores for an application.

Table 2. A list of anomalies and their configurations.

Anomaly type	Configuration
cpuoccupy	-u 100%, 80%
cachecopy	-c L1,-m 1 / -c L2 -m 2
membw	-s 4K, 8K, 32K
memleak	-s 1M, -p 0.2 / -s 3M -p 0.4 / -s 10M -p 1

4.3 Implementation Details

We fill out missing metric values with linear interpolation because some metric values can be missing while telemetry data is being collected. We also take the difference of cumulative counters in every step because we care about the increase, not the raw value. We strip out the first and last 60 seconds of the collected time-series data to prevent fluctuations during the initialization and termination phases of applications. We experiment with 45 and 60-second windows since we want to provide effective results while minimizing the delay. In the end, we choose 60-second windowing since it led to a better F1-score, anomaly miss rate, and false alarm rate during evaluation.

We use the "Min-Max" scaler on training data and scale the test data using the same scaler during the model training to minimize possible data leakage. After the training, the scaler is saved as a Python *pickle* object, and the same scaler is used during runtime diagnosis. We split the dataset into 5-folds and iteratively fit the model on the remaining folds while holding the remaining one for validation. While splitting the data, we use stratified sampling where the class distribution matches with the whole dataset in each fold.

We experiment with Random Forest and Gradient Boosting Machines. Random Forest is composed of multiple decision trees, and it generally combines the average of each decision tree or applies majority voting during prediction. We use the implementation in *scikit-learn* that uses majority voting for the classification tasks [28]. Gradient Boosting Machines are decision-tree-based classifiers and use gradient boosting, which produces a prediction result using an ensemble of weak prediction models. We use *Extreme Gradient Boosting (XGBoost)* [14] and *Light Gradient Boosting Machine (LGBM)* [19] implementations. Even though XGBoost and LGBM are part of the gradient boosting machines, they have different techniques while splitting the nodes. We use the LGBM as a final classifier due to high performance in F1-score, anomaly miss rate, and false alarm rate.

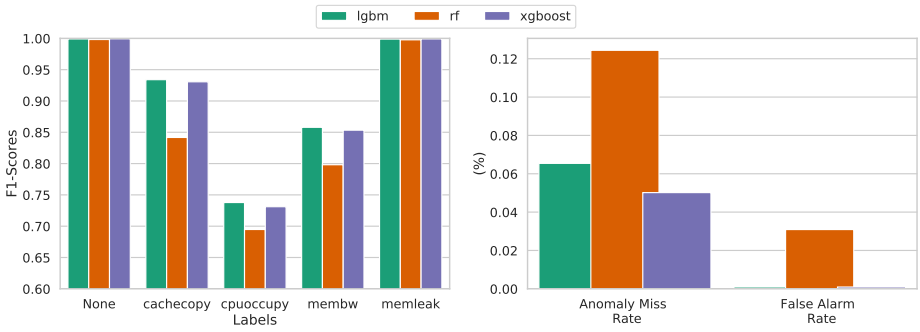


Fig. 3. Macro average F1-scores of LGBM, Random Forest, and XGBoost models. LGBM and XGBoost perform up to 10% better in cachecopy, cpuoccupy, and membw anomalies than Random Forest. LGBM and XGBoost are 2x better than Random Forest in anomaly miss rate while achieving near zero false alarm rate.

5 Evaluation

We evaluate our model under 3 different experimental scenarios and report F1-score, false alarm rate (i.e., false-positive rate), and anomaly miss rate (i.e., false-negative rate). F1-score is defined as the harmonic mean of precision and recall, where precision shows what percentage of positive class predictions were correct and recall shows what percentage of actual positive class samples were identified correctly. Eq. 1 shows the false alarm rate, which corresponds to the percentage of normal runs identified as one of the anomaly types. Eq. 2 shows the anomaly miss rate, which corresponds to the percentage of anomalous runs (any anomaly) identified as normal.

$$False\ Alarm\ Rate = \frac{False\ Positives}{False\ Positives + True\ Negatives} \tag{1}$$

$$Anomaly\ Miss\ Rate = \frac{False\ Negatives}{False\ Negatives + True\ Positives} \tag{2}$$

5.1 Anomaly Diagnosis Scores

We present anomaly diagnosis results for each anomaly type with anomaly miss rate and false alarm rate in Fig. 3. Average F1-scores are 0.91, 0.90, and 0.87, for LGBM, XGBoost, and Random Forest, respectively. All models achieve an almost perfect diagnosis F1-score for windows without anomalies. LGBM and XGBoost outperform Random Forest in terms of F1-score in all cases. It is expected to see a similar performance among XGBoost and LGBM since they are fundamentally similar. XGBoost and LGBM miss 0.05% of anomalous windows and achieve almost zero false alarm rates. The F1-scores of *cpuoccupy* anomaly are lower than other anomalies because all classifiers confuse *cpuoccupy* with *membw* anomaly due to similar CPU utilization characteristics.

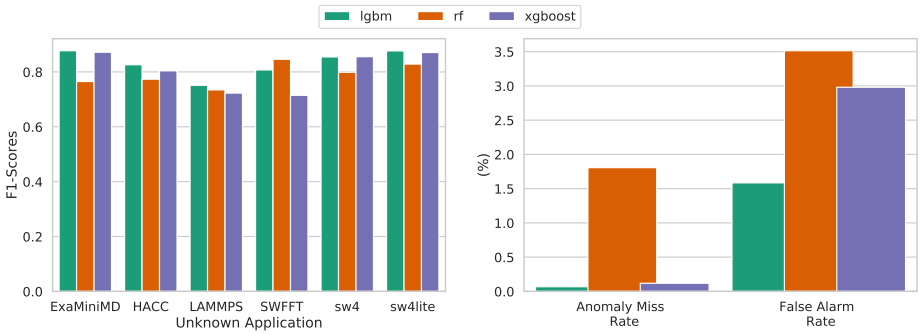


Fig. 4. Macro average F1-score of models when an unknown application exists in the test set. LGBM is 8% and 4% better than Random Forest and XGBoost in F1-score on average, respectively. LGBM and XGBoost achieve lower false alarm and anomaly miss rates than Random Forest.

5.2 Unknown Applications

In a production system scenario, it is likely to encounter applications that do not exist in the training data, so we evaluate the model's performance with scenarios where unknown applications exist in the test data while keeping all the anomalies. First, we remove all runs of the selected application from the training set and then include only the removed application to the test set. We repeat this setup for each application and report average F1-scores, anomaly miss rate, and false alarm rate in Fig. 4. Except for SWFFT, XGBoost and LGBM are up to 10% better than Random Forest in F1-scores, and LGBM is the best performing one, including for anomaly miss rate and false alarm rate.

5.3 Unknown Application Inputs

Another common scenario in production systems is running the same application with different input decks. In our dataset, we have three input sizes (small, medium, large) for each application, and each input size corresponds to the different number of compute nodes we run the application. We evaluate the model's performance with scenarios where unknown application inputs exist in test data. First, we remove all runs of the selected input size from the training set and then include only the removed input size in the test set. We repeat this setup for each input size and report average F1-scores along with anomaly miss rate and false alarm rate in Fig. 5. For all unknown input types, LGBM and XGBoost can diagnose anomalies with F1-scores over 0.75. LGBM is the most robust one to unknown input sizes in terms of anomaly miss rate and false alarm rate.

5.4 Discussion on Deployment

At SNL, HPCMON hosts the analysis and visualization pipeline for understanding HPC system data. The pipeline uses the Scalable Object Store (SOS)

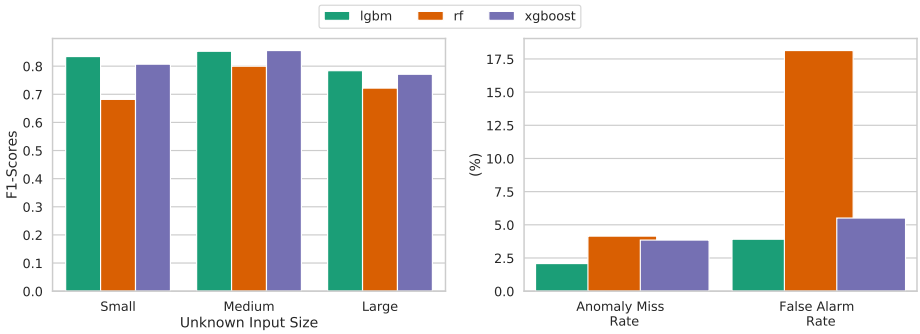
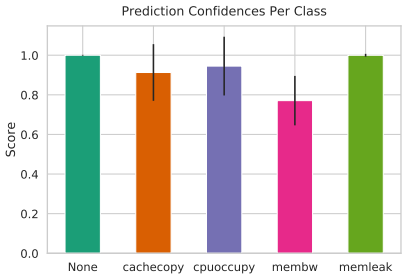


Fig. 5. Macro average F1-score of models when an unknown application input exists in the test set. LGBM and XGBoost are 13% better than Random Forest in F1-scores. While models have comparable anomaly miss rates, LGBM is 8x, and XGBoost is 4x better than Random Forest in false alarm rate.

database that has unique indexing to efficiently manage structured data, including time series [30]. The first advantage of SOS is that we can query the data of interest instead of getting the whole data and selecting afterward. This enables E2EWatch to provide anomaly diagnosis results without waiting for the completion of an application. The second advantage is that SOS enables easy configuration for user-derived metrics, hence supports a flexible analysis development cycle. This feature enables the easier development of new models for different classification tasks as new needs arise. At a high level, *E2EWatch* requires the following components to provide diagnosis results at runtime in another production system:

1. **Monitoring framework** that can collect numeric telemetry data from compute nodes while applications are running. Even though we only experiment with LDMS, it can be adapted to other popular monitoring frameworks such as Ganglia [25], Examon [7] by modifying the wrappers in the data collection phase.
2. **Labeled data** that is composed of anomalous and normal compute node telemetry data. It is possible to create labeled data sets using a suite of applications and synthetic anomalies. Another option is to use telemetry data labeled by users.
3. **Backend web service** that can provide telemetry data on the fly to the trained model. We use the existing Django web application deployed on the monitoring server [30]. It is possible to use other backend web services that can handle client requests and query data from the database. If runtime diagnosis is not necessary, it is also possible to run the pickled model after the application run is completed.

We calculate the model's prediction confidence in the test data for correctly classified samples and provide the statistical distribution in Fig. 6. Prediction confidences are also necessary to monitor possible *concept drift* and *data drift*.



Confidence Level Translation Table

Calculation	Confidence
$Val \leq \mu - 3\sigma$	None
$\mu - 3\sigma < Val \leq \mu - 2\sigma$	Low
$\mu - 2\sigma < Val \leq \mu - \sigma$	Medium
$\mu - \sigma < Val \leq \mu + \sigma$	High
$\mu + \sigma < Val$	Very High

Fig. 6. The deployed model's prediction confidence for each anomaly type. Error bars show one standard deviation above and below. Except *membw* anomaly, the model has very high confidence for each class. The confidence translation table provides a way for a user to interpret prediction results easily.

Concept drift happens when the statistical properties of the target variable change, e.g., some anomalies might start showing different characteristics. Data drift occurs when the statistical properties of streaming data change. Especially in HPC systems, seasonality and user trends could change according to usage, e.g., conference deadlines and periodic system upgrades. For example, suppose we observe a sudden decrease in prediction confidences of samples predicted *normal*. In that case, it can point out a possible drift scenario.

We implement two filtering techniques to increase robustness against false alarms and anomaly misses for runtime anomaly diagnosis. The first one is *consecutive filtering*, where we keep the original prediction label if it persists in C consecutive windows; otherwise, we replace it with *the normal* label. Even though this approach reduces false alarms, it increases the anomaly miss rate since we replace anomalies directly with the normal label. The second one is *majority filtering*, where we replace the original prediction label with the most frequent class label in C consecutive windows. Majority filtering generally reduces false alarms

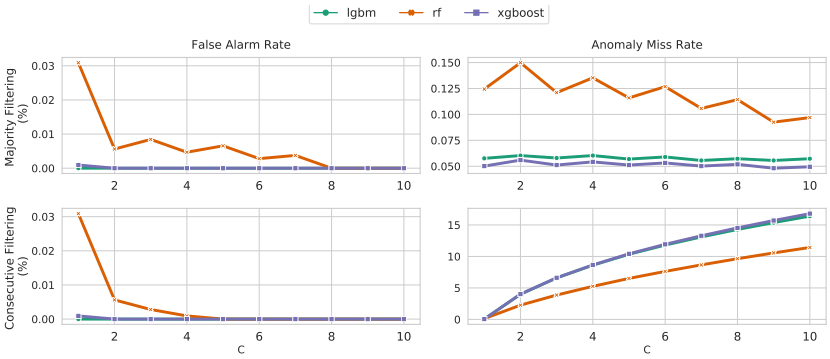


Fig. 7. The first row shows majority filtering results, and the second row shows consecutive filtering results for different C values. LGBM and XGBoost maintains a constant false alarm rate in both filtering techniques, whereas Random Forest's false alarm rate is reduced almost three times.

and anomaly miss rates. As an example, using the following window predictions, [memleak, memleak, membw, memleak, memleak], majority filtering will return [memleak,memleak, memleak,memleak,memleak], whereas consecutive filtering will return [normal,normal,normal,memleak,memleak] when C equals 3. In Fig. 7, we show the effect of these two filtering techniques on all classifiers. While consecutive filtering increases anomaly miss rate in all classifiers, it significantly reduces Random Forest's false alarm rate. On the other hand, LGBM and XGBoost have almost constant false alarm rates and anomaly miss rates with both techniques.

6 Conclusion and Future Work

Automatic and online diagnosis of performance anomalies has been increasingly important as we move towards the Exascale computing era; however, transitioning from research to real-world applications is challenging. In this paper, we demonstrated *E2EWatch*, an end-to-end anomaly diagnosis framework, on a 1488-node HPC production machine. *E2EWatch* provides job and node-level visualizations in an easy-to-interpret dashboard. Our future work includes creating a data set that is composed of a large set of popular HPC applications and evaluating the framework with a scenario where the scale of the number of applications and input decks is comparable to production systems. The second direction is to conduct user studies and evaluate the performance of our framework in a real-world setting. The third direction we plan to explore is deployment process for production systems that have compute nodes with GPUs.

Acknowledgment

This work has been partially funded by Sandia National Laboratories. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under Contract DE-NA0003525. This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

References

1. Agelastos, A., Allan, B., Brandt, J., Cassella, P., Enos, J., Fullop, J., Gentile, A., Monk, S., Naksinehaboon, N., Ogden, J., et al.: The lightweight distributed metric service: a scalable infrastructure for continuous monitoring of large scale computing systems and applications. In: Proc. of the Int. Conf. for High Performance Computing, Networking, Storage and Analysis (SC). pp. 154–165 (2014)
2. Agelastos, A., Allan, B., Brandt, J., Gentile, A., Lefantzi, S., Monk, S., Ogden, J., Rajan, M., Stevenson, J.: Toward rapid understanding of production HPC applications and systems. In: IEEE International Conf. on Cluster Computing. pp. 464–473 (2015)

3. Ahad, R., Chan, E., Santos, A.: Toward autonomic cloud: Automatic anomaly detection and resolution. In: International Conf. on Cloud and Autonomic Computing. pp. 200–203 (2015)
4. Arzani, B., Ciraci, S., Loo, B.T., Schuster, A., Outhred, G.: Taking the blame game out of data centers operations with netpoirot. In: Proceedings of the ACM SIGCOMM Conference. pp. 440–453 (2016)
5. Ates, E., Tuncer, O., Turk, A., Leung, V.J., Brandt, J., Egele, M., Coskun, A.K.: Taxonomist: Application detection through rich monitoring data. In: European Conference on Parallel Processing. pp. 92–105. Springer (2018)
6. Ates, E., Zhang, Y., Aksar, B., et al.: Hpas: An hpc performance anomaly suite for reproducing performance variations. In: Proceedings of the 48th Intl. Conference on Parallel Processing. p. 1–10. ACM (Aug 2019)
7. Bartolini, A., Borghesi, A., Libri, A., Beneventi, F., Gregori, D., Tinti, S., Gianfreda, C., Altoè, P.: The davide big-data-powered fine-grain power and performance monitoring support. In: Proceedings of the 15th ACM Int. Conf. on Computing Frontiers. pp. 303–308 (2018)
8. Bhatele, A., Mohror, K., Langer, S.H., Isaacs, K.E.: There goes the neighborhood: performance degradation due to nearby jobs. In: SC'13: Proceedings of the Int. Conf. on High Performance Computing, Networking, Storage and Analysis. pp. 1–12 (2013)
9. Bhatele, A., Thiagarajan, J.J., Groves, T., Anirudh, R., Smith, S.A., Cook, B., Lowenthal, D.K.: The case of performance variability on dragonfly-based systems. In: IEEE International Parallel and Distributed Processing Symposium (IPDPS). pp. 896–905 (2020)
10. Bhuyan, M.H., Bhattacharyya, D., Kalita, J.K.: Nado: network anomaly detection using outlier approach. In: Proceedings of the Int. Conf. on Communication, Computing & Security. pp. 531–536 (2011)
11. Borghesi, A., Bartolini, A., Lombardi, M., Milano, M., Benini, L.: A semisupervised autoencoder-based approach for anomaly detection in high performance computing systems. Engineering Applications of Artificial Intelligence **85**, 634–644 (2019)
12. Bourassa, N., Johnson, W., Broughton, J., Carter, D.M., Joy, S., Vitti, R., Seto, P.: Operational data analytics: Optimizing the national energy research scientific computing center cooling systems. In: Proceedings of the 48th Int. Conf. on Parallel Processing: Workshops. pp. 1–7 (2019)
13. Brandt, J.M., DeBonis, D., Gentile, A.C., Lujan, J., Martin, C., Martinez, D.J., Olivier, S.L., Pedretti, K., Taerat, N., Velarde, R.: Enabling advanced operational analysis through multi-subsystem data integration on trinity. Tech. rep., Sandia National Lab.(SNL-CA), Livermore, CA (United States) (2015)
14. Chen, T., Guestrin, C.: Xgboost: A scalable tree boosting system. In: Proceedings of the ACM Int. Conf. on Knowledge Discovery and Data Mining. pp. 785–794 (2016)
15. Dalmazo, B.L., Vilela, J.P., Simoes, P., Curado, M.: Expedite feature extraction for enhanced cloud anomaly detection. In: IEEE/IFIP Network Operations and Management Symposium. pp. 1215–1220 (2016)
16. Das, A., Mueller, F., Rountree, B.: Aarohi: Making real-time node failure prediction feasible. In: 2020 IEEE Int. Parallel and Distributed Processing Symposium (IPDPS). pp. 1092–1101 (2020)
17. Dorier, M., Antoniu, G., Ross, R., Kimpe, D., Ibrahim, S.: Calciom: Mitigating i/o interference in hpc systems through cross-application coordination. In: IEEE International Parallel and Distributed Processing Symposium. pp. 155–164 (2014)

18. Jayathilaka, H., Krintz, C., Wolski, R.: Performance monitoring and root cause analysis for cloud-hosted web applications. In: Proceedings of the 26th International Conference on World Wide Web. pp. 469–478 (2017)
19. Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., Liu, T.Y.: Lightgbm: A highly efficient gradient boosting decision tree. Advances in neural information processing systems **30**, 3146–3154 (2017)
20. Klinkenberg, J., Terboven, C., Lankes, S., Müller, M.S.: Data mining-based analysis of hpc center operations. In: IEEE Int. Conference on Cluster Computing (CLUSTER). pp. 766–773 (2017)
21. Lan, Z., Zheng, Z., Li, Y.: Toward automated anomaly identification in large-scale systems. IEEE Trans. on Parallel and Distributed Systems **21**(2), 174–187 (2009)
22. Leung, V.J., Bender, M.A., Bunde, D.P., Phillips, C.A.: Algorithmic support for commodity-based parallel computing systems. Tech. rep., Sandia National Laboratories (2003)
23. Marathe, A., Zhang, Y., Blanks, G., Kumbhare, N., Abdulla, G., Rountree, B.: An empirical survey of performance and energy efficiency variation on intel processors. In: Proceedings of the 5th Int. Workshop on Energy Efficient Supercomputing. pp. 1–8 (2017)
24. Massey Jr, F.J.: The kolmogorov-smirnov test for goodness of fit. Journal of the American statistical Association **46**(253), 68–78 (1951)
25. Massie, M.L., Chun, B.N., Culler, D.E.: The ganglia distributed monitoring system: design, implementation, and experience. Parallel Computing **30**(7), 817–840 (2004)
26. Nair, V., Raul, A., Khanduja, S., Bahirwani, V., Shao, Q., Sellamanickam, S., Keerthi, S., Herbert, S., Dhulipalla, S.: Learning a hierarchical monitoring system for detecting and diagnosing service issues. In: Proc. of the ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining. pp. 2029–2038 (2015)
27. Netti, A., Müller, M., Guillen, C., Ott, M., Tafani, D., Ozer, G., Schulz, M.: Dcdb wintermute: enabling online and holistic operational data analytics on hpc systems. In: Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing. pp. 101–112 (2020)
28. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
29. Sandia National Laboratories: HPC capacity cluster platforms, https://hpc.sandia.gov/HPC%20Production%20Clusters/index.html
30. Schwaller, B., Tucker, N., Tucker, T., Allan, B., Brandt, J.: HPC system data pipeline to enable meaningful insights through analysis-driven visualizations. In: IEEE Int. Conf. on Cluster Computing (CLUSTER). pp. 433–441 (2020)
31. Shaykhislamov, D., Voevodin, V.: An approach for dynamic detection of inefficient supercomputer applications. Procedia Computer Science **136**, 35–43 (2018)
32. Skinner, D., Kramer, W.: Understanding the causes of performance variability in HPC workloads. In: Proceedings of the IEEE Workload Characterization Symposium. pp. 137–149 (2005)
33. Tuncer, O., Ates, E., Zhang, Y., Turk, A., Brandt, J., Leung, V.J., Egele, M., Coskun, A.K.: Online diagnosis of performance variation in HPC systems using machine learning. IEEE Trans. on Parallel and Distributed Systems **30**(4), 883–896 (2018)
34. Xie, C., Xu, W., Mueller, K.: A visual analytics framework for the detection of anomalous call stack trees in high performance computing applications. IEEE transactions on visualization and computer graphics **25**(1), 215–224 (2018)

35. Zasadziński, M., Muntés-Mulero, V., Solé, M., Carrera, D., Ludwig, T.: Early termination of failed hpc jobs through machine and deep learning. In: European Conference on Parallel Processing. pp. 163–177. Springer (2018)
36. Zhang, X., Meng, F., Chen, P., Xu, J.: Taskinsight: A fine-grained performance anomaly detection and problem locating system. In: IEEE Int. Conf. on Cloud Computing (CLOUD). pp. 917–920 (2016)
37. Zhang, Y., Groves, T., Cook, B., Wright, N.J., Coskun, A.K.: Quantifying the impact of network congestion on application performance and network metrics. In: IEEE International Conference on Cluster Computing (CLUSTER). pp. 162–168 (2020)