Interpretable Anomaly Detection for Monitoring of High Performance Computing Systems

Elisabeth Baseman
lissa@lanl.gov
USRC[1]
Los Alamos National Laboratory
Los Alamos, NM 87544

Sean Blanchard
seanb@lanl.gov
USRC[1]
Los Alamos National Laboratory
Los Alamos, NM 87544

Nathan DeBardeleben
ndebard@lanl.gov
USRC[1]
Los Alamos National Laboratory
Los Alamos, NM 87544

Amanda Bonnie
noranzyk@lanl.gov
HPC Environments
Los Alamos National Laboratory
Los Alamos, NM 87544

Adam Morrow[2]
adampmorrow@gmail.com
College of Physical and Mathematical Sciences
Brigham Young University
Provo, UT 84604

ABSTRACT

The Trinity supercomputer at Los Alamos National Laboratory collects data from a large number of sensors, producing a massive stream of data on the order of terabytes per day. Previous supercomputers relied on humans skimming this data for incorrect behaviors, but this approach is not feasible with current machines, or future exascale class machines, which will produce approximately 25 times more data than Trinity. Instead, an automated system that can scale to the large data stream will be required. We present Interpretable and Interactive Classifier-Adjusted Density Estimation with Temporal components (iCADET), an extension of classifier-adjusted density estimation, for anomaly detection and exploratory data analysis. Our methods take advantage of the interpretability of random forest classifiers to present explanations to an analyst regarding why the algorithm ranks certain data points as more likely to be anomalous. We also include temporal relational features in our analysis, and provide a mechanism for analysts to modify the anomaly detection method in real-time in order to silence uninteresting anomalies. We find that iCADET successfully recovers a variety of synthetic anomalies injected into the Trinity supercomputer System Environment Data Collections (SEDC) data set. We consult domain experts to confirm that our methods uncover behaviors in the un-altered data set that are informative for systems analysts and troubleshooters.

CCS Concepts

•**Computing methodologies** → **Anomaly detection;** *Semi-supervised learning settings;* •**Computer systems organization** → *Maintainability and maintenance;*

[1]This work was performed at the Ultrascale Systems Research Center (USRC) at Los Alamos National Laboratory, supported by the U.S. Department of Energy contract DE-AC52-06NA25396. The publication has been assigned the LANL identifier LA-UR-16-23740.

[2]This work was performed during an internship at the Ultrascale Systems Research Center at Los Alamos National Laboratory in Los Alamos, NM.

Keywords

anomaly detection, density estimation, high performance computing, exploratory analysis, explanatory analysis

1. INTRODUCTION

Large high performance computing (HPC) facilities collect data sampled at a high rate from a wide variety of sensors in order to continuously monitor the health of the system. These sensors generate massive quantities of data, far too large for human analysts to sift through. However, information about the health, performance, and energy usage of an HPC system may be hidden in this monitoring sensor data, currently collected at a rate on the order of terabytes per day. This information is critical to the correct operation of current, and more importantly, future exascale class machines. The HPC community expects exascale machines to be roughly 25 times more performant, and therefore produce 25 times more data, than Trinity. Therefore, we must automate the process of sifting through this massive data set, presenting only the most potentially interesting information to human analysts, while filtering out uninteresting data. Due to the complex nature of modern supercomputers, it is exceedingly difficult to discover root causes of incorrect behavior without detailed, time consuming analysis. In order to streamline root cause discovery, human analysts need to understand *why* an anomaly detection algorithm might mark a certain sensor reading as potentially problematic. In addition, an analyst does not want to be bothered with too many false alarms, wasting time he could be spending caring for the supercomputing facility, which will be especially true for future exascale machines. This points to an immediate need for interpretable, interactive anomaly detection on sensor time series data. In this work, we present a pipeline of methods that achieve this goal.

We examine data collected from the Trinity supercomputer during open science testing at Los Alamos National Laboratory (LANL). In this configuration, known as "Phase 1 Trinity", the machine was a 9,436 node Cray XC30. Phase 1 Trinity nodes contain two Haswell Xeon E5-2698v3 16 core processors. Trinity Phase 2, expected in the coming months, will consist of an equal number of Intel Xeon Phi Knights Landing processors. Each Phase 1 node has 128 GB of main memory and a Cray Aries-based network. Trinity is water cooled with chilled doors on each cabinet and will draw 9MW of power in its final configuration. Cray's sensor infrastructure collects data on hardware behavior including: voltages and power consumption of components, temperatures, fan speeds, and both air and water temperatures. Cray's System Environment Data Collections (SEDC) tool polled data once per second during the open science runs. Several of these sensor data streams are expected to alter their values depending on loads imposed by user applications. They are also expected to vary during different computation phases of any particular application. Other sensor readings are expected to stay steady independent of the load on the machine. Sensors reading outside known ranges or varying at unexpected times could be indicative of a serious error that could negatively impact the short term behavior of the Trinity computer as well as its long term health.

This paper makes the following contributions:

- **Interpretable, Interactive, Classifier-Adjusted Density Estimation in Time (iCADET)**: an extension of classifier-adjusted density estimation [6], with explanatory, interactive, and temporal components.

- A Python tool which implements our methods and can be directly applied to sensor data from high performance computing systems.

The rest of this paper proceeds as follows: Section 2 reviews related work on anomaly detection and machine learning in the context of high performance computing. Section 3 describes our working definition of anomalous, while Section 4 details the available sensor data from the Trinity supercomputer. Section 5 explains our four-step method, including feature extraction, density estimation, interpretation, and interaction. Section 6 describes our experimental setup for evaluation of our anomaly detector and Section 7 presents our results. Finally, Section 8 describes future work on handing streaming, online data, and Section 9 concludes.

2. RELATED WORK

Significant previous work exists related to machine learning for computing and monitoring applications. High performance computing systems are so large that they can no longer be considered deterministic systems, and researchers are moving towards treating data from computing systems as observational data appropriate for statistical modeling.

2.1 Anomaly Detection for HPC Applications

Several recent studies focus on discovering interesting and anomalous behaviors in the field of computer system monitoring, particularly in detecting unusual performance [12].

Dueño *et al.* provided an R package for clustering and feature exploration in HPC data, but lacked available data to determine the effectiveness of their approach [4]. Work by Florez *et al.* at Mississippi State University focused on detecting anomalous program execution by generating application profiles and applying neural networks and hidden Markov models [5]. Peiris *et al.* proposed a tool that visualizes performance counters such that developers could manually specify threshold values to define anomalous behavior [19]. We aim to automatically detect anomalies, rather than relying on user-specified, hard-coded thresholds.

Several studies by Song Fu *et al.* at the University of North Texas applied time series-based anomaly detection techniques to data from cloud computing infrastructures. This work used most relevant principal component analysis and wavelet analysis to detect anomalous system behavior in real-time [10, 18]. However, their approach did not consider a large number of heterogeneous data sources, and focused on data collected from hardware performance counters and virtual machine performance metrics.

Finally, work by Xu *et al.* at the University of California Berkeley leveraged system console logs for detecting system-level problems [21, 22, 23]. They developed a multi-stage process, involving first pre-processing and parsing the console log messages, filtering out "uninteresting" events, and applying principal component analysis, to detect unusual behaviors. While this approach was successful, Xu *et al.* only applied their methods to filesystems, focusing on anomalous file openings and closures. Our task is to detect anomalous behavior in real-time on a much larger scale.

2.2 General Learning for HPC Applications

In addition to anomaly detection, the field of applying more general machine learning algorithms to data from computing systems, especially for analyzing computing system log data, has been growing in recent years. Although the following studies do not focus on anomaly detection, they investigate several methods and tasks related to our work.

2.2.1 *Analysis of Monitoring Logs*

Studies by Fulp *et al.* and Gainaru *et al.* both attempted to identify and predict interesting system events using sliding time windows and by characterizing frequencies of meaningful log messages [7, 8]. Fulp *et al.* used support vector machines trained on log messages from a Linux-based compute cluster to predict hard drive failures with 73% accuracy approximately 2 days in advance of failures. Their features were constructed by extracting signatures of log messages within a time window, based on how frequently each message type appeared within the message grouping. They noted that SVMs provide the ability to investigate internal weight functions, and therefore feature importances, but did not take all steps to turn their approach into a general explanatory/interpretable algorithm. Meanwhile, Gainaru *et al.* took a very similar approach in their feature extraction, then attempted to use their log message frequencies to identify correlated log events. They found many correlations, but lacked enough data to interpret their findings.

Work at Microsoft Research by Mickens *et al.* used interactive decision trees to identify possible misconfigurations of user-level applications [15]. They allowed humans to stay "in the loop," asking for feedback while building decision trees. This approach is similar to our ultimate goal of including domain experts and analysts in the diagnosis and mitigation of anomalous system behavior, but remains a supervised problem relying on labeled data (and as such will not function as an anomaly detector).

Sabato *et al.* at IBM proposed a tool for ranking system logging messages by usefulness to the user [20]. Their method of clustering machines based on use cases and then ranking event log messages based on likelihood performed very well in terms of feedback from users regarding importance of highly ranked messages. This anomaly detection strategy is similar to the approach we take in our work, but, unlike our work, did not incorporate a temporal component or a variety of input data from monitoring sensors.

2.2.2 *Intelligent System Scheduling*

Researchers have also applied machine learning techniques to the design of intelligent schedulers for computer systems. Heuristic-based machine learning and data mining have been used for operating system process scheduling on both distributed and single-processor systems with some success [1]. In addition, work by Moss *et al.* at the University of Massachusetts Amherst investigated using reinforcement learning with hand-tuned features for local instruction scheduling, with notable success [2, 16, 14]. While methods similar to those found useful for machine schedulers may also apply in the HPC monitoring domain, the particulars of these applications are not pertinent to our specific task.

3. DEFINING "INTERESTING"

In general, we define an anomaly to be a single data point in the context of a larger data set that is drawn from a different generative process than the rest of the data. However, while a data point may technically and statistically be an anomaly, it may not be an *interesting* anomaly. For example, when monitoring network traffic a new printer being installed on the network may show up as a statistical anomaly, but probably is not particularly interesting to an analyst searching for malicious behaviors. Our analysts care about anomalous sensor data which indicates that they can apply an intervention or mitigation strategy to the computing system. This includes anomalies that indicate problems with the sensors themselves or the recording of their data.

For example, certain kinds of sensors, such as those measuring aspects of CPU behavior, are expected to vary with the computational load (CPUs under load will draw more power at higher voltage and see an increase in temperature). Network chips under I/O load behave similarly. Trinity is designed to keep temperature as stable as possible across the machine, a feature that differs from previous supercomputing facilities, so significant temperature variations by location within the machine would be unexpected and indicative of a problem. This is true of the Trinity machine due to its being water cooled, as opposed to previous machines that were air cooled. In addition, each sensor has minimum and maximum ratings that related hardware components should not exceed. As these hardware components age, analysts will want to analyze whether sensors begin to show readings outside of normal ranges, an effect which may happen at different rates depending on the particular hardware component. Finally, unexpected sensor readings may correlate with aspects of user behavior and workloads, informing more intelligent adaptations to jobs in the workload queue.

Because of the wide variety of potentially interesting but difficult to uncover behaviors, we involve domain experts in our evaluation of the quality of the anomalies our methods find. We allow the analyst to silence specific kinds of anomalies through manipulation of the anomaly detection mechanism itself (rather than requiring a human to generate and input domain knowledge). In this way, we use a malleable definition of "anomalous," combining strict statistical definitions with expert human knowledge.

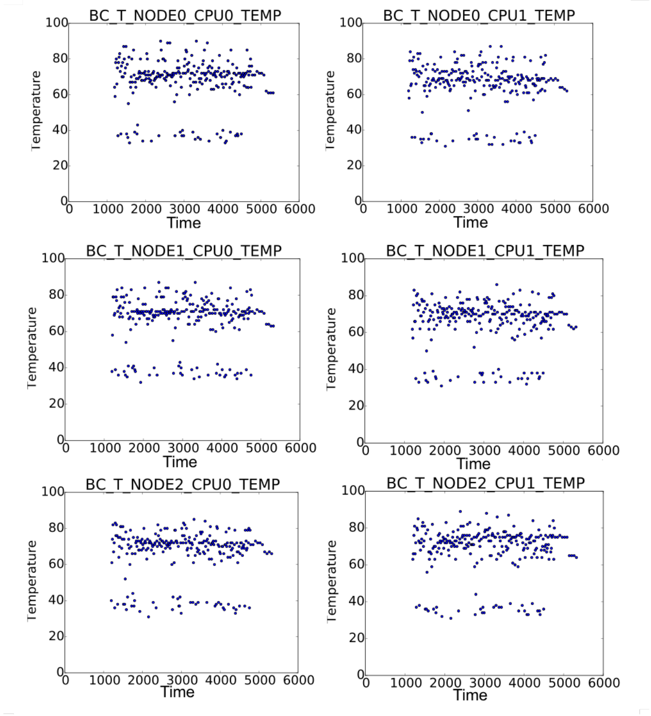

Figure 1: Example temperature sensor data from two CPUs on each of two nodes, over one day. This is a very small subset of the available time series data for the entire Trinity supercomputer. Note that for these sensors in particular, normal behavior appears to belong to two distinct temperature groupings.

4. SYSTEM MONITORING SENSOR DATA

HPC systems include a variety of sensors. These sensors typically measure environmental conditions such as distributed voltages, power consumption, and temperatures. Sensors may measure conditions on both global and local scales, such as the temperature of the machine room versus the temperature on a particular CPU, as well as scales in between. For this work we examine local conditions within each Cray blade on the Trinity supercomputer. Cray's SEDC (System Environment Data Collections) utility provides information on a large number of environmental conditions, including voltages, power consumption, air speed, and temperature of CPUs, air, and water [3]. We focus on one specific set of temperature sensors, and more data sources will be included in future analysis.

For this work, we focus our efforts on the Cray blade CPU temperature sensors, although our methods can be easily applied to any set of HPC sensors. We focus our attention on these particular sensors because they have multiple behavioral modes that are considered normal, as shown in Figure 1. Figure 1 shows example CPU temperature data over the course of one day of computation on Trinity, clearly depicting two normal modes of behavior – one when the CPU is under load and the other when it is not. An anomalous temperature reading on any one of the CPUs that does not fall into either of the acceptable behavior modes, may indicate a serious problem with the overall system health.

Additional interest in studying CPU temperatures is due to Trinity's new water cooling system, the first of its kind to be installed at LANL. This system uses warmer than industry standard water exchanges to cool the exhaust heat from each rack of nodes on the machine. Previous air-cooled designs required over-cooling of the machine room to ensure that CPUs would not overheat. This, combined with the lower thermal conductivity of air, led to serious inefficiencies in the operation of the data centers in which LANL supercomputers resided. However, operating in this new cooling regime requires thorough monitoring of the system so that LANL operations staff can be certain that the machine remains within design specifications.

Future analysis of SEDC sensors may confound readings of CPU temperatures with their closely related power draw sensors. Power consumption on future supercomputers will be one of the major challenges as machines approach exaflop performance. Future machines will not be able to operate with all nodes drawing maximum power simultaneously, but will instead rely on sets of nodes running in lower power modes while others run in higher ones. Being able to accurately and quickly find nodes that are not behaving as expected during an application run will not only prevent overheating, but also will be required to prevent brown-outs due to overdrawing power.

5. APPROACH

The two primary goals of our work are: anomaly detection, and exploratory analysis of the Trinity supercomputer sensor data. Our approach expands upon previous work in classifier-adjusted density estimation [6], extending it to Interpretable and Interactive Classifier-Adjusted Density Estimation in Time (iCADET). iCADET relies on a four-stage process: feature extraction, density estimation, interpretation, and interaction with the analyst. Figure 2 shows the iCADET pipeline at a high level, and the following sections give more detail about each step in the iCADET process.

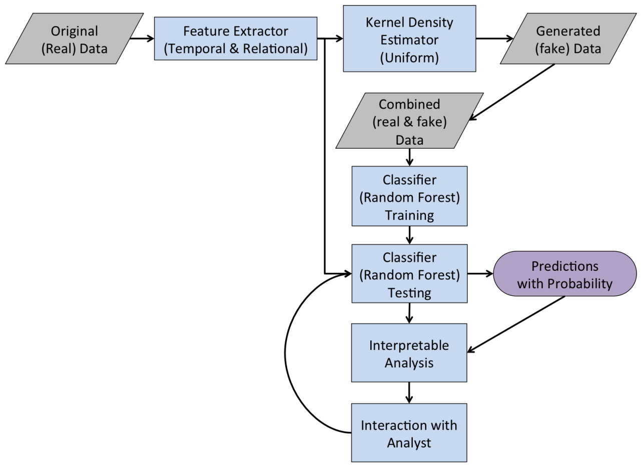
Original (Real) Data
Feature Extractor (Temporal & Relational)
Kernel Density Estimator (Uniform)
Generated (fake) Data
Combined (real & fake) Data
Classifier (Random Forest) Training
Classifier (Random Forest) Testing
Predictions with Probability
Interpretable Analysis
Interaction with Analyst

Figure 2: The iCADET pipeline. Expanding upon classifer-adjusted density estimation, we include a relational temporal feature extraction step, explanatory/interpretable analysis, and provide a mechanism for the analyst to modify the anomaly detection decision-making online.

5.1 Feature Extraction (Stage 1)

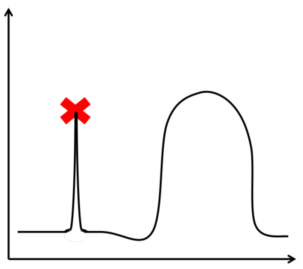

Figure 3: Extracting time-dependent features, such as gradients, from our data allows us to catch anomalies such as the one shown in red. The actual values of the features at the anomalous point are not unusual due to the range covered by the rest of the data. However, the faster change to reach the feature value at the anomalous point allows us to identify it as anomalous.

Supercomputing sensor data, due to the variety of sensors, is inherently inhomogeneous and temporal. Because of this, we modify our anomaly detection method to automatically extract temporal relational features from the sensor data [9, 17]. We extract feature gradients at each time stamp and include these as additional features. We expect gradient values to be particularly indicative of anomalous behavior, especially when used in combination with raw feature values. As shown in Figure 3, including gradients allows us to uncover certain kinds of anomalous behaviors that raw feature values cannot identify, such as spikes that do not reach an unusual value, but that are interesting because of an unexpected jump.

5.2 Density Estimation (Stage 2)

After extracting features from the Trinity sensor data, we apply density estimation, with the ultimate goal of ranking the data based on degree of "anomalousness"'. For our density estimation procedure, we diverge from the usual method of density estimation, instead applying a combination of uniform density estimation and probabilistic classification [11].

Classifier-adjusted density estimation (CADE) is based on the observation that a uniform density estimate combined with probabilistic output of a classifier results in a highly accurate density estimator [6]. Figure 4 shows the high-level algorithm structure. The method works by first estimating the density of a data set, for which a simple uniform distribution works well. Next, this density estimator generates a synthetic data set with the same number of data points as the original (real) data. A classifier which outputs probabilities rather than a single class label, a random forest in this case, is trained on a combined data set of both real and fake data points to discriminate between the two. Then, the real data is run through the classifier and ranked by probability of being drawn from the fake data set. This gives a ranking of real data points in terms of "anomalousness".

5.3 Interpretation (Stage 3)

We build upon the CADE technology to make our anomaly detection results human-interpretable. An analyst needs to understand *why* the algorithm classifies certain data points as anomalous and not others. iCADET allows an analyst

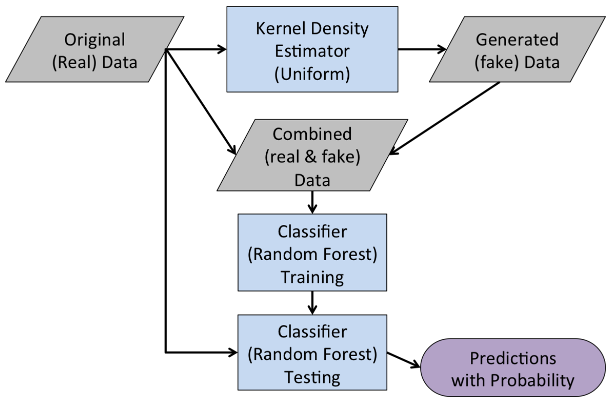
Original (Real) Data
Kernel Density Estimator (Uniform)
Generated (fake) Data
Combined (real & fake) Data
Classifier (Random Forest) Training
Classifier (Random Forest) Testing
Predictions with Probability

Figure 4: The classifier-adjusted density estimation (CADE) algorithm: (1) a uniform density estimate generates a fake data set of the same size as the real data set. (2) A random forest classifier learns to distinguish between the real and fake data, reporting probabilities. (3) The trained random forest reports probabilities for the original real data. (4) The real data is ranked based on those probabilities, which are a proxy for "anomalousness."

to dig deeper into an anomaly of interest, walking back up each decision tree in the random forest which classified the particular data point as anomalous. In this way, an analyst can explore the potential reasons behind the anomaly.

iCADET accomplishes this exploration by first identifying the set of decision trees within the random forest that classified the given data point as anomalous. For each of these trees, it finds the leaf corresponding to the given data point, as shown in Figure 5. Figure 6 demonstrates how starting at this leaf, iCADET begins to walk back up the decision tree, taking note of each decision node where a difference in a single feature value would have resulted in a classification as non-anomalous — i.e., investigating possible counterfactuals. At each decision node where this is the case, our method remembers the relevant decision rule. iCADET collects each relevant decision rule it finds in each decision tree, condenses them into as few rules as possible, and then reports back to the analyst a list of rules that caused the algorithm to classify the given point as anomalous. Essentially, iCADET present the analyst with a list of features and thresholds that, had the feature value been different with respect to the threshold value, the point would have been considered normal.

For example, if iCADET finds that three trees in the random forest classify a given data point as anomalous, the relevant rules might be: '`volts` $> 1.5 \Rightarrow$ anomalous', '`temp` $> 80 \Rightarrow$ anomalous', and '`temp` $> 90 \Rightarrow$ anomalous'. Our algorithm searches through these potential new rules and consolidates any rules regarding the same feature, with the same inequality direction, into a single rule. If a feature always appears in the potential new rules as less than some threshold, the potential rule suggested to the analyst takes the mininmum of all thresholds found, and similarly if the feature is always greater than some threshold. In this way, given the example rules just mentioned, the method would suggest the rules '`volts` $> 1.5 \Rightarrow$ anomalous' and '`temp` $> 90 \Rightarrow$ anomalous' to the analyst. In the interpreta-

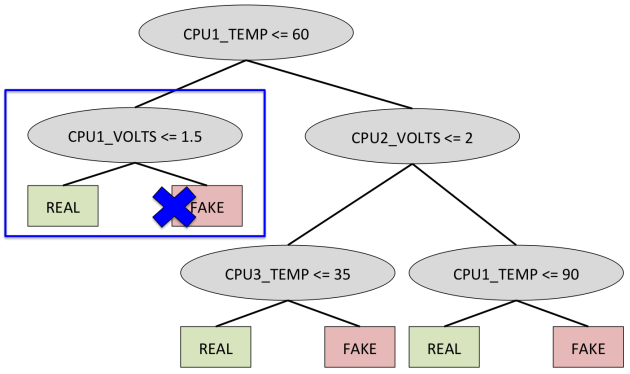

Figure 5: Given a particular data point that iCADET has classified as likely anomalous (shown as the blue X), we loop over all decision trees in the random forest that contributed to this decision. For each tree, we then identify the relevant subtree that resulted in a label of "anomalous", identified here inside the blue box.

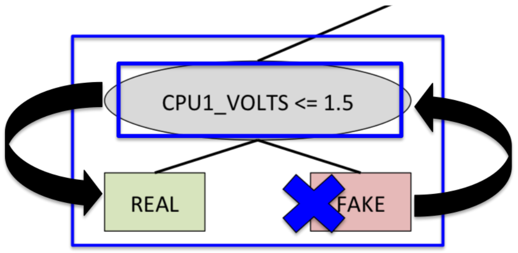

Figure 6: After identifying the leaf resulting in a classification of "anomalous" (shown as the blue x), iCADET climbs back up the decision tree, checking the other children of the decision nodes to find feature thresholds that, had the particular feature value been slightly different, would have resulted in a non-anomalous classification. In this example, we walk one hop up the decision tree, notice that the left child is a leaf indicating non-anomalous data, and report the rule at the related decision node.

tion step of iCADET, the analyst is then given the option to throw out uninteresting suggested rules and prune the decision trees, so that uninteresting anomalies can be silenced.

5.4 Interaction (Stage 4)

After presenting the user with a list of potential rules for classifying a data point as an interesting anomaly, iCADET requests feedback from the analyst. It prompts the analyst to indicate which rules, if any, should be discarded. In the case where the analyst indicates a rule should be discarded (probably due to it distinguishing statistical anomalies which are not interesting for the analyst's particular task), iCADET allows the analyst to modify and prune the decision trees within the random forest, as shown in Figure 7. When an analyst indicates a suggested rule is not useful, our methods consolidate that decision node into a single leaf, with all data points reaching that leaf being labeled normal and thus not triggering an alert. Then, the random forest has been modified so that the analyst will no longer waste

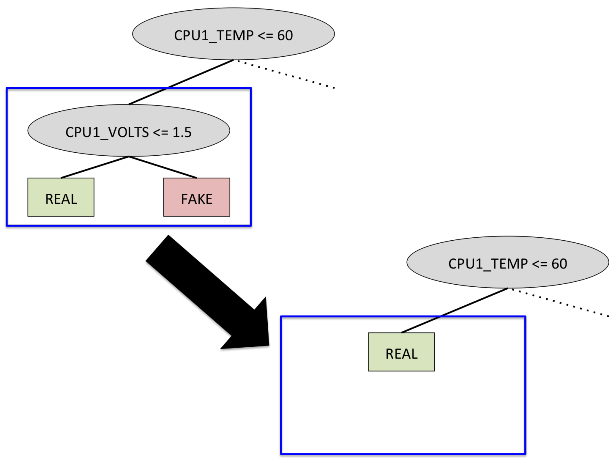

Figure 7: When an analyst indicates that a particular rule is not useful for distinguishing interesting anomalies, such as 'CPU1_VOLTS ≤ 1.5', we consolidate the relevant decision node in each decision tree within the random forest to be a single leaf, with all data points being assigned a non-anomalous label.

time silencing the same uninteresting alerts and can focus energy on exploring more interesting and relevant data.

6. EXPERIMENTAL SETUP

We focus our analysis on the Trinity blade controller CPU temperature sensors, because anomalies in these measurements could indicate serious problems with the overall health of the system. To both quantitatively and qualitatively test the performance of our anomaly detector, we report the results of injecting synthetic anomalies into the available time series, as well as reporting qualitative evaluations obtained from systems monitoring domain experts.

For our quantitative analysis, we inject anomalous points with varying feature values into the actual Trinity SEDC data, as well as varying the number of injected anomalies. Results from varying the number of anomalies will be useful when we extend our technique to streaming data – if something within the computing system goes bad, we may suddenly get a deluge of anomalous data points and we want to still be able to detect this. We vary the number of injected anomalies from 1 to 100, in a Trinity SEDC data set of approximately 6,000 data points.

Along with varying the number of anomalies, we also vary the number of features within the anomaly that take on unusual values. Because there are a total of 24 features, including temporal features, we look at having 1, 3, 6, 12, and all 24 features take on unusual values.

When constructing an anomaly, we define six distinct kinds of synthetic anomalies, which we term 'outside max', 'outside min,' 'feature max', 'feature min', 'outside mixed', and 'feature mixed.' 'Outside' refers to injecting a feature value that falls outside the max or min value for that feature observed in the real data, while 'feature' refers to inserting the max or min observed feature value from the rest of the data. For 'outside max,' we set all feature values to be greater than the maximum observed value for that respective feature in the rest of the data set. For 'outside min,' we set all feature values to be less than the minimum observed value. Similarly, for 'feature max,' we set each feature value to be the maximum observed value for that feature in the rest of the data, and likewise for 'feature min.' 'Outside mixed' injected anomalies alternate by feature whether the injected value is greater than the observed max or less than the observed min, and 'feature mixed' refers to alternately injecting the maximum or minimum observed feature value. The kinds of upward and downward spikes in sensor values simulated by these synthetic anomalies are the kinds of behaviors systems monitoring analysts would like to be able to find, as these spikes can indicate over- or under-worked hardware as well as faulty hardware or architectures and more general problems with system health.

6.1 Preprocessing

While all features of the SEDC sensor data are numeric, we find that at some points in time the sensors were faulty, resulting in missing feature values. Because we are interested in anomaly detection, we replace any missing values with the relevant feature average value. In this way, we are able to use all valid feature values in our data, and essentially make any missing data negligible with respect to our anomaly detection method. In some cases, we find broken sensors that did not report any values, and ignore them. Overall, the amount of missing data was negligible.

Due to the large volume of our data, we sample down to a more manageable data set size to obtain a proof of concept of our methods. We sample the time series at 15-second intervals for one day, resulting in approximately 6,000 data points for each of 12 temperature sensors. This then becomes a total of 24 features after extracting temporal information.

6.2 Evaluation

We present both quantitative and qualitative results. For each combination of number of injected anomalies, number of anomalous features, and type of anomaly, we report the probability score of the highest-ranked injected anomaly as well as precision at 5. However, because all data points that would be considered statistically anomalous are not necessarily interesting to an analyst (a new printer on a network may appear as a statistical anomaly but is a boring data point), we report on the usefulness of our tool and the anomalous points it finds in the real data, as qualitatively evaluated by a Trinity supercomputer systems analyst at Los Alamos National Laboratory.

7. RESULTS

7.1 Quantitative

Table 1 shows a representative subset of our results from experiments using injected anomalies with feature values outside the actual data range. The "top score" column indicates the likelihood of the most anomalous data point, as calculated by the random forest classifier. We also report precision at 5 for each experiment. Table 2 shows similar results for the anomalies injected using real feature values. We find near perfect precision at 5 for most experiments (an average precision at 5 of 87%), which is more than sufficient for useful data exploration. We also find that the next-highest ranked anomalous points are those close to the injected anomalies in time, because of the temporal features that we include. In other words, a spike in one sensor value may result in two alerts — one when the spike occurs and

Anomaly Type	% of Data	Number of Features	Top Score	Precision at 5
outside max	0.02 %	24	0.3917	1.0
outside max	1.66 %	24	0.0833	0.6
outside max	0.02 %	12	0.4167	1.0
outside max	1.66 %	12	0.3542	0.4
outside max	0.02 %	6	0.3375	1.0
outside max	1.66 %	6	0.0917	1.0
outside max	0.02 %	3	0.4000	1.0
outside max	1.66 %	3	0.2500	1.0
outside max	0.02 %	1	0.3792	1.0
outside max	1.66 %	1	0.2083	1.0
outside min	0.02 %	24	0.3917	1.0
outside mixed	0.02 %	24	0.3708	1.0
outside mixed	1.66 %	24	0.05	0.4
outside mixed	0.02 %	12	0.3792	1.0
outside mixed	1.66 %	12	0.1417	0.2
outside mixed	0.02 %	6	0.3042	1.0
outside mixed	1.66 %	6	0.0917	0.8
outside mixed	0.02 %	3	0.3708	1.0
outside mixed	1.66 %	3	0.1417	1.0

Table 1: Representative quantitative results for injected anomalies using feature values above and/or below the values seen in real data. Note the high precision at 5 results, indicating that our methods are useful for exploratory purposes.

another if the value returns to normal. This is reasonable behavior because many sensors have multiple normal behavior modes, and alerting on both changes lets the analyst have a more detailed picture of the sensor behavior.

Furthermore, the interpretable step of our method successfully identifies the rules used to inject anomalies, suggesting new rules such as ‘BC_T_NODE2_CPU0_TEMP > 91.5’ when the anomalous feature value was set to 100.0, or ‘BC_T_NODE0_CPU1_TEMP < 33.4’ when the anomalous feature value was the feature’s global minimum of 30.0.

7.2 Qualitative

According to a Trinity systems analyst from Los Alamos National Laboratory, all CPU temperatures are expected to appear very similar, with a few behavior modes corresponding to idle, normal workloads, and heavy workloads. Chips are designed to slow themselves down if they begin to approach the maximum temperature that they can withstand. Interestingly, iCADET tags data points close to the maximum allowable temperature as anomalous, which would alert an analyst to workloads that are potentially too heavy for the system. In addition, iCADET tags some middle-ground data points, with temperatures that do not fall into any expected behavioral mode, particularly values in the room-temperature range. This gives the analyst crucial information that the CPUs registering these values may be broken, or less worryingly, extremely underused.

8. TOWARDS ONLINE METHODS

While iCADET is already a useful exploratory tool in high performance computing settings, we would like to extend our method to run in real-time on extremely massive data sets. We project that sensor data, as well as data from other sources such as system logs, will be coming off of future supercomputers at a rate on the order of 25 times that analyzed in our current work.

Anomaly Type	% of Data	Number of Features	Top Score	Precision at 5
feature max	0.02 %	24	0.1417	1.0
feature max	1.66 %	24	0.0042	0.6
feature max	0.02 %	12	0.2083	1.0
feature max	1.66 %	12	0.0083	0.4
feature max	0.02 %	6	0.3708	1.0
feature max	1.66 %	6	0.2333	0.6
feature max	0.02 %	3	0.3917	1.0
feature max	1.66 %	3	0.1417	1.0
feature max	0.02 %	1	0.3	1.0
feature max	1.66 %	1	0.275	1.0
feature min	0.02 %	24	0.1792	1.0
feature min	1.66 %	24	0.0083	0.2
feature min	0.02 %	12	0.0042	1.0
feature min	1.66 %	12	0.0042	1.0
feature min	0.02 %	6	0.3708	1.0
feature min	1.66 %	6	0.0917	0.6
feature min	0.02 %	3	0.3875	1.0
feature min	1.66 %	3	0.1625	1.0
feature min	0.02 %	1	0.3958	1.0
feature min	1.66 %	1	0.2167	1.0
feature mixed	0.02 %	24	0.1583	1.0
feature mixed	0.02 %	12	0.2167	1.0

Table 2: Representative quantitative results for injected anomalies using feature values drawn from the maximum and minimum values seen in real data. Again note the high precision results.

Fortunately, there is already significant interest in applying probabilistic machine learning techniques to streaming data [13]. We plan to leverage this and similar work to adjust our current methods in terms of feature extraction and density estimation, including a storage strategy pared down to only essential features for anomaly detection in our particular high performance computing domain.

9. CONCLUSIONS

We have presented iCADET, an anomaly detection and data exploration method specifically for high performance computing sensor data. The method involves a four-step process of relational temporal feature extraction, classifier-adjusted density estimation (CADE), interpretable analysis of anomalies, and integration of interactive analyst feedback. We find that our methods easily recover anomalies injected into the Trinity supercomputer open science data set with an average precision at 5 of 87%, and find informative anomalous behavior in the un-altered data set as well. In addition, the interpretable analysis in iCADET successfully recovers the rules used to inject synthetic anomalies. While iCADET can be applied to any temporal data set, it is particularly useful for the sensor data obtained from high performance computing facilities, and we find that its exploratory function is extremely useful for systems analysts. In the future, we plan to augment our techniques with strategies for handling larger volumes of data in a streaming setting.

10. REFERENCES

[1] G. Anderson, T. Marwala, and F. V. Nelwamondo. Use of data mining in scheduler optimization. *arXiv preprint arXiv:1011.1735*, 2010.

[2] J. Cavazos and J. E. B. Moss. Inducing heuristics to decide whether to schedule. *ACM SIGPLAN Notices*, 39(6):183–194, 2004.

[3] Cray. *System Environment Data Collections Guide*. Cray Publications.

[4] S. J. Q. Dueño, E. A. Sáez, Y. M. C. Bonaparte, and H. Kettani. Detecting anomalies for high performance computing resilience. In *Computer and Automation Engineering (ICCAE), 2010 The 2nd International Conference on*, volume 4, pages 5–7. IEEE, 2010.

[5] G. Florez-Larrahondo, Z. Liu, Y. S. Dandass, S. M. Bridges, and R. Vaughn. Integrating intelligent anomaly detection agents into distributed monitoring systems. *Journal of Information Assurance and Security*, 1(1):59–77, 2006.

[6] L. Friedland, A. Gentzel, and D. Jensen. Classifier-adjusted density estimation for anomaly detection and one-class classification. SIAM, 2014.

[7] E. W. Fulp, G. A. Fink, and J. N. Haack. Predicting computer system failures using support vector machines. *WASL*, 8:5–5, 2008.

[8] A. Gainaru, F. Cappello, J. Fullop, S. Trausan-Matu, and W. Kramer. Adaptive event prediction strategy with dynamic time window for large-scale hpc systems. In *Managing Large-scale Systems via the Analysis of System Logs and the Application of Machine Learning Techniques*, page 4. ACM, 2011.

[9] L. Getoor. *Introduction to statistical relational learning*. MIT press, 2007.

[10] Q. Guan and S. Fu. Adaptive anomaly identification by exploring metric subspace in cloud computing infrastructures. In *Reliable Distributed Systems (SRDS), 2013 IEEE 32nd International Symposium on*, pages 205–214. IEEE, 2013.

[11] K. Hempstalk, E. Frank, and I. H. Witten. One-class classification by combining density and class probability estimation. In *Machine Learning and Knowledge Discovery in Databases*, pages 505–519. Springer, 2008.

[12] O. Ibidunmoye, F. Hernández-Rodriguez, and E. Elmroth. Performance anomaly detection and bottleneck identification. *ACM Computing Surveys (CSUR)*, 48(1):4, 2015.

[13] T. Jayram, A. McGregor, S. Muthukrishnan, and E. Vee. Estimating statistical aggregates on probabilistic data streams. *ACM Transactions on Database Systems (TODS)*, 33(4):26, 2008.

[14] A. McGovern and J. E. B. Moss. Scheduling straight-line code using reinforcement learning and rollouts. In M. J. Kearns, S. A. Solla, and D. A. Cohn, editors, *Advances in Neural Information Processing Systems 11*, pages 903–909. MIT Press, 1999.

[15] J. Mickens, M. Szummer, and D. Narayanan. Snitch: Interactive decision trees for troubleshooting misconfigurations. In *In Proceedings of the 2007 Workshop on Tackling Computer Systems Problems with Machine Learning Techniques*, 2007.

[16] J. E. B. Moss, P. E. Utgoff, J. Cavazos, D. Precup, D. Stefanovic, C. E. Brodley, and D. Scheeff. Learning to schedule straight-line code. In M. I. Jordan, M. J. Kearns, and S. A. Solla, editors, *Advances in Neural Information Processing Systems 10*, pages 929–935. MIT Press, 1998.

[17] J. Neville, D. Jensen, L. Friedland, and M. Hay. Learning relational probability trees. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 625–630. ACM, 2003.

[18] H. S. Pannu, J. Liu, and S. Fu. Aad: Adaptive anomaly detection system for cloud computing infrastructures. In *Reliable Distributed Systems (SRDS), 2012 IEEE 31st Symposium on*, pages 396–397. IEEE, 2012.

[19] M. Peiris, J. H. Hill, J. Thelin, S. Bykov, G. Kliot, and C. Konig. Pad: Performance anomaly detection in multi-server distributed systems. In *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*, pages 769–776. IEEE, 2014.

[20] S. Sabato, E. Yom-Tov, A. Tsherniak, and S. Rosset. Analyzing system logs: A new view of whatâĂŹs important. In *Proceedings of the 2nd USENIX workshop on Tackling computer systems problems with machine learning techniques*, pages 1–7. USENIX Association, 2007.

[21] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan. Online system problem detection by mining patterns of console logs. In *Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on*, pages 588–597. IEEE, 2009.

[22] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan. Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 117–132. ACM, 2009.

[23] W. Xu, L. Huang, and M. I. Jordan. Experience mining google's production console logs. In *SLAML*, 2010.