Signal Processing Based Method for Real-Time Anomaly Detection in High-Performance Computing

Arunavo Dey, Tanzima Islam, Chase Phelps
Department of Computer Science
Texas State University
San Marcos, TX 78666
hcs77,tanzima,chaseleif@txstate.edu

Christopher Kelly
Brookhaven National Laboratory
Computer Science Initiative
Long Island, NY
ckelly@bnl.gov

***Abstract*—Performance anomalies can manifest as irregular execution times or abnormal execution events for many reasons, including network congestion and resource contention. Detecting such anomalies in real-time by analyzing the details of performance traces at scale is impractical due to the sheer volume of data High-Performance Computing (HPC) applications produce. In this paper, we propose formulating HPC performance anomaly detection as a signal-processing problem where anomalies can be treated as noise. We evaluate our proposed method in comparison with two other commonly used anomaly detection techniques of varying complexity based on their detection accuracy and scalability. Since real-time in-situ anomaly detection at a large scale requires lightweight methods that can handle a large volume of streaming data, we find that our proposed method provides the best trade-off. We then implement the proposed method in CHIMBUKO, the first online, distributed, and scalable workflow-level performance trace analysis framework. We also propose a novel metric for comparing our proposed signal-based anomaly detection algorithm with two other methods—a function of accuracy, F1 score, and detection overhead. Our experiments demonstrate that our proposed approach achieves a 99% improvement for the benchmark datasets and a 93% improvement with CHIMBUKO traces using the proposed metric.**

***Index Terms*—Real-time anomaly detection in HPC, Signal based anomaly detection, Fast Fourier Transform, CHIMBUKO.**

I. INTRODUCTION

Many modern HPC applications are comprised of tightly-coupled components which execute concurrently, exchange information, and compete for hardware resources [6]. Contention for shared resources such as compute, memory, I/O, and network resources can introduce a large fluctuation in performance, otherwise known as performance anomalies. Detection of performance anomalies can aid in better utilization of resources, improving the performance of applications, and reducing the cost of running large-scale applications.

Due to complexities in communication patterns and resource contention, assessing performance and identifying possible bottlenecks in large-scale jobs requires tools capable of capturing these interactions while avoiding impacting application performance. CHIMBUKO [11] is the first online, distributed, and scalable workflow-level performance trace analysis framework that detects performance anomalies in **function execution time** at scale. However, anomalous behavior can also manifest outside of execution time. For instance, an anomaly that may also affect the under-utilization of a system is a considerable variation in the number of instructions across ranks. Anomalies such as this appear in the number of instructions and are captured by a hardware performance counter. Extending CHIMBUKO's existing capability to include an analysis of performance counters can be challenging. It requires re-designing the framework's data structures to hold more information while maintaining a small memory footprint.

Currently, CHIMBUKO uses the Histogram-based Outlier Score (HBOS) algorithm [8] to detect anomalies in function execution time with a server-client execution model where the server and the clients periodically synchronize. HBOS creates histogram bins to reflect statistically normal accumulated function execution times. A challenge in extending HBOS to detect anomalies in an arbitrary number of counters is that hundreds of bins could be generated per counter, where each bin needs to be described using several parameters. The algorithm's parameters, information needed to duplicate histograms of all counters, is synchronized between the server and client components via the network. Using HBOS for hundreds of counters could easily create bottlenecks in memory usage and network traffic as the number of monitored counters increases.

To address this challenge, we propose a new method for handling a large volume of performance counter information arriving in real-time and detecting anomalies in their values. Specifically, we propose formulation of performance anomaly detection in HPC as a noise detection problem akin to that in the area of signal processing. This novel formulation enables leveraging the principled approach of noise detection using Fast Fourier Transform (FFT)-based signal decomposition. In a real-time analysis framework, the large volume of counter values can quickly accumulate to an unmanageable size. To reduce the memory footprint of client processes and the

volume of data transferred to the server, each client aggregates counter values. In contrast, the server uses statistical metrics to build a signal across ranks as a secondary metric. Clients use these secondary metrics to label individual counters as anomalous if they are beyond a configurable range.

We compare our proposed approach with two other algorithms of varying complexity from statistical analysis and Machine Learning (ML) domains. Since real-time anomaly detection requires methods to be fast *and* accurate, we propose a new metric to evaluate algorithms to assess their suitability in real-time performance anomaly detection scenarios. Specifically, the metric calculates the trade-off between algorithmic performance and the overhead introduced using $\frac{accuracy \times F1score}{overhead}$, where a higher value indicates more algorithmic performance in less time, and that an algorithm is more suitable for real-time use.

In summary, the main contributions of this paper are:

- Formulating performance anomaly detection as a signal processing problem to use FFT to identify anomalies.
- Proposing a novel metric for comparing the applicability of anomaly detection techniques in real-time performance monitoring frameworks.
- A comparison of the proposed approach with different algorithms using benchmark applications and trace data from a scientific application for anomaly detection based on the proposed metric.
- We demonstrate that our proposed approach provides the best trade-off between accuracy and F1 score, ideal for real-time anomaly detection.

In this paper, we compare our proposed FFT-based approach with Auto Regressive Integrated Moving Average (ARIMA) and Robust Random Cut Forest (RRCF). ARIMA [22] is a widely used statistical method, whereas RRCF [9] is a more recent ML based anomaly detection technique. Our thorough evaluations show that the $accuracy \times F1score$ of the FFT-based method on six benchmark applications offers as much as 99.1% improvement compared to ARIMA, 99.4% compared to RRCF, and, for performance traces from CHIMBUKO, an improvement of up to 90% compared to ARIMA and 95% with RRCF.

The rest of this paper is organized as follows: in Section II, we describe the CHIMBUKO framework, the HBOS algorithm, currently implemented by CHIMBUKO, and the ARIMA and RRCF algorithms; Section III describes our proposed approach of a signal processing-based method for performance anomaly detection; Section V describes the experimental setup; Section VI describes the evaluation results; we present related work in Section VII; we conclude the paper in Section VIII.

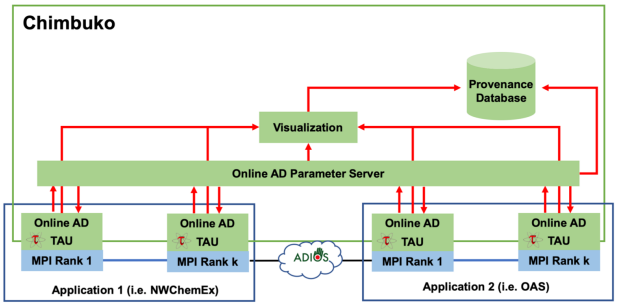


Fig. 1: Overview of CHIMBUKO

II. BACKGROUND

A. CHIMBUKO *framework*

Figure 1 provides an overview of the CHIMBUKO framework. The CHIMBUKO framework provides functionality through three components: Anomaly Detection Component (AD), Parameter Server Component (PS), and Provenance Database (ProvDB). The AD component consists of a single process per node which runs alongside the application processes of a workflow. The nodes running an AD instance and workflow processes are called "body nodes". Each AD instance receives performance data from Tuning and Analysis Utilities (TAU) [1] and determines if a function is anomalous by applying the HBOS algorithm to the function's execution time. The PS and the ProvDB components each run as a single multi-threaded process and are typically run on the "head-node", separate from the body nodes. The ProvDB records anomaly provenance information and the PS coordinates synchronization of calculated runtime statistics among all AD instances.

B. *HBOS algorithm*

Each AD component creates a histogram of the elapsed run time for all executions of each function. The PS component aggregates each client's histogram for each function, adjusting the number of bins and bin widths as necessary, and returns an aggregated histogram for each function to each AD component. The HBOS algorithm has configurable parameters, including the $hbos\_threshold$ to control the algorithm's sensitivity, and the maximum number of bins of a histogram which affects the algorithm's tolerance to variance.

Additionally, to provide information to provenance, the CHIMBUKO framework records information of each histogram including the location of bin edges, widths, and the count of values within each bin. For each function, the AD calculates a score for each bin within the function's histogram using the equation: $score = -1 * \lg(contribution + \alpha)$, where $contribution$ is the number of values within the bin divided by the number of values within the histogram, and α is defined as a near-zero value, a floating-point epsilon. The parameter $hbos\_threshold$, set to 0.99, is used to calculate $l\_threshold$ using the equation $l\_threshold = min\_score + (hbos\_threshold * (max\_score - min\_score))$,

where min_score and max_score are, respectively, the minimum and maximum of bin scores.

HBOS determines a function to be anomalous if it contains an event whose run time is beyond the edges of the function's outer left- or right-most histogram bins by more than 5% of the width of the bins of the histogram, or if the function's histogram contains a bin whose score is greater than $l\_threshold$.

C. ARIMA

ARIMA is a regression-based ML model [22], [5], that predicts the current observation using previous observations. If the current observation Y_t depends on the past observations $Y_{t-1}, Y_{t-2}, ...$, then ARIMA predicts the expected value of the current observation Y_t using Equation 1:

$$Y_t = (\beta_1 + \beta_2) + (\phi_1 Y_{t-1} + \phi_2 Y_{t-2} + ... + \phi_p Y_{t-p}) + (\omega_1 \epsilon_{t-1} + \omega_2 \epsilon_{t-2} + ... + \omega_p \epsilon_{t-q}) \quad (1)$$

In this equation, β_1 and β_2 are constants, p is the number of lagged observations included in the calculation, and q is the size of the moving window. The "I" in ARIMA stands for "Integrated", highlighting that ARIMA uses the differences in the observations for predictions.

D. Robust Random Cut Forest (RRCF)

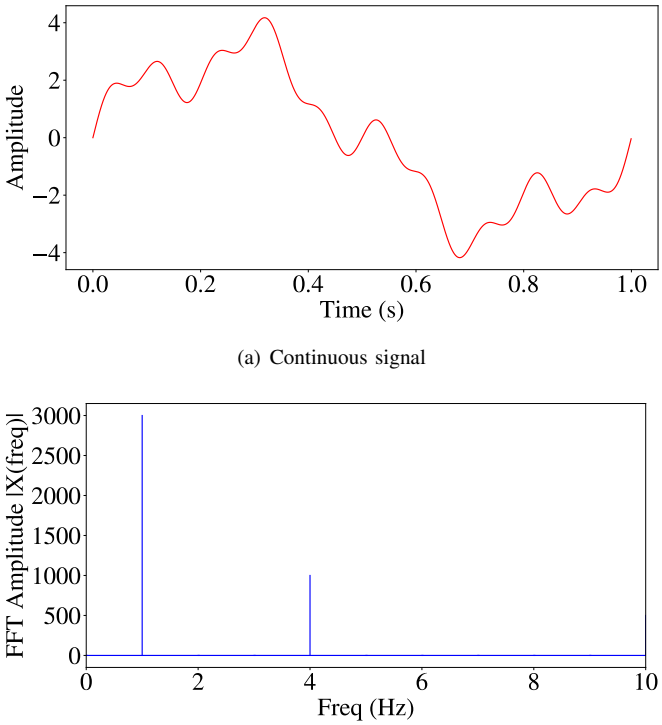
RRCF [9] is a tree-based ensemble method that is designed to handle high-dimensional streaming data. In RRCF, performance counter values at each timestep are inserted as nodes in a tree. Each node of the tree has a bit depth equal to the number of bits needed to store the value. The model's complexity is the sum of the bit depths of all nodes already inserted. The expected change in complexity, or displacement, caused by the insertion of node x is quantified by Equation 2, where x is the newly inserted node, Z is the set of all other nodes, T is the RRCF tree, and $f(y, Z, T)$ represents the depth of y in T defined by all other nodes in Z. The term on the right-hand side denotes the expected change in the bit depths of all leaves in the tree when x is inserted.

$$Disp(x, Z) = \sum_{T, y \in Z - x} Pr[T](f(y, Z, T) - f(y, Z - x, T)) \quad (2)$$

A new value is considered an outlier if the insertion of that value significantly increases the complexity of the model as a new node in the tree.

RRCF calculates the displacement x by forming "collusive displacement", where a set of colluders, C, along with the point of interest, x, are removed. C is the set of duplicates and near duplicates that could mask the presence of outliers. Equation 3 describes the calculation of collusive displacement:

$$CoDisp(x, Z, |S|) = \underset{S \subseteq Z, T}{E}[\max_{x \in C \subseteq S} \frac{1}{|C|} \sum_{y \in S - C} (f(y, S, T) - f(y, S - C, T''))] \quad (3)$$



(a) Continuous signal

(b) Transformed discrete signal from FFT

Fig. 2: FFT transformation of a Continuous Signal

III. Our Approach

A. Signal Processing based Anomaly Detection

With inspiration from signal processing, we formulate the problem of performance anomaly detection similar to noise detection in analog signals. To consider performance analysis as noise detection, we form a signal by concatenating counter values per function across ranks for all call paths over each time step as a vector. Similar to signal processing, we then apply the FFT method to obtain frequency and amplitude components from continuous counter signals. Figure 2 illustrates this transformation, where the continuous signal in Figure 2(a) is discretized as shown in Figure 2(b). An anomalous counter value is either a large or small amplitude value whose change in variance compared to previous timesteps is greater than a threshold. The rationale for using a method such as FFT is that values from continuous variables, such as execution time across ranks and timesteps can be transformed into a discrete form. Such discretization eliminates the need for calculating histogram bin widths, which can be ad-hoc in the HBOS algorithm. We aggregate counter values for several timesteps by averaging them to create a continuous signal before we check for performance anomalies.

Each workflow process applies FFT to a series of counter values and, using the real parts of the signal, calculates their

mean using Equation 4.

$$\mu = \sum_{i=1}^{n} [x_i/n] \tag{4}$$

Likewise, the variance of the real values is calculated by Equation 5.

$$\sigma^2 = \sum_{i=1}^{n} [(x_i - \mu)^2/n] \tag{5}$$

If there are no historical statistics for a counter, we set the historical mean, $\mu_h = \mu$, the historical variance, $\sigma_h^2 = \sigma^2$, the historical difference, $diff_h = 0$, and do not mark the counter as anomalous.

We use the parameters: $threshold = 3$, $\alpha = 0.8$, $\beta = 42$, and $\gamma = 0.3$. When historical statistics exist, we calculate the change in variance as $\delta = |\sigma_h^2 - \sigma^2|/\sigma_h^2$, and, if not $\delta > threshold$, we mark the counter as anomalous and assign $\delta = (\gamma * \delta) + ((1 - \gamma) * \delta_h)$. We then assign $\delta_h = \delta$ and $\mu_h = (\alpha * \mu_h) + ((1-\alpha) * \mu)$. Concluding the evaluation of each counter, we set $\sigma_h^2 = (\beta * \sigma^2) + ((1-\beta) * \sigma_h^2)$ if we had marked the counter as anomalous, or $\sigma_h^2 = (\alpha * \sigma^2) + ((1 - \alpha) * \sigma_h^2)$ otherwise.

IV. Integration with Chimbuko

To integrate our proposed anomaly detection algorithms for counter-wise and rank-wise anomaly detection in Chimbuko, we extend both the AD and PS components described in Section II. Each AD component receives lists of execution data, where execution data includes function names, call paths, unique function IDs, event entry and exit timestamps, and all counter values collected during the execution of each function. Each AD component determines functions to be anomalous according to the chosen anomaly detection algorithm which is responsible for analysis of the incoming stream of execution data. The AD component sends provenance information of anomalous events, including the rank ID and function description, to the ProvDB component.

For each additional algorithm in this work, we design a map of values keyed according to unique call path and counter in the AD component. During synchronization, AD components send these data structures to the PS component which then merges these values and returns the merged structure to each AD.

In the FFT algorithm, each AD additionally appends counter values to a list during iteration of the execution data. Following iteration of events in each function, if we have a minimum number of values for a counter, we perform the FFT algorithm and determine whether a counter is anomalous as described in Section III, clear the counter value list, and update our historical statistics. Historical statistics of each PS are merged using the minimum of all AD statistics.

V. Experimental Setup

In this section, we describe the setup of experiments and metrics we use to evaluate the accuracy of the proposed anomaly detection algorithms using trace data of the NWChem application collected by the Chimbuko framework. To evaluate the performance of each algorithm, we augment the trace data with manually injected anomalies. We run Chimbuko to replay the trace of NWChem for each new algorithm and use a pseudo-random number generator with a fixed seed to generate artificial counter values to be treated as anomalies.

During each time step, if the distance between the generated artificial value and the mean is greater than $\sigma\times$ standard deviations, then it is inserted as an anomalous value. The total number and position of injected anomalies are chosen using the pseudo-random number generator.

A. Applications:

NWChem: NWchem is a high-performance computational chemistry tool that provides quantum chemical and molecular dynamics functionality. It is designed to run on HPC systems, workstations, and clusters, and scales both in terms of problem size and resource usage.

Numenta Anomaly Benchmark (NAB): The Numenta Anomaly Benchmark (NAB) [14] dataset is a corpus of 58 real-time series data. This benchmark contains both real-world and artificially generated data. We list below the datasets we use for evaluation along with a brief description of each:
Twitter: A collection of real-world Twitter data where each value represents the number of mentions of large publicly-traded companies every 5 minutes.
Temperature: Ambient temperature data from an office.
Ad_exchange: Rate of clicks for online advertisements.
EC2 utilization: CPU usage data collected as AWS server metrics by the AmazonCloudwatch service.
CPU utilization: Average CPU usage data from Amazon Web Services from a cluster.
Sedov: A Sedov solver simulation from the FLASH application.

B. Evaluation Metrics:

Table I summarizes the metrics that we use for performance evaluation. In addition to the metrics listed in Table I, we use our proposed metric $(accuracy \times F1\ score)/overhead$ to compare each algorithm.

VI. Results

In this section, we compare the effectiveness of our proposed FFT-based method, ARIMA, and RRCF methods in anomaly detection of the applications described in Section V. Specifically, we investigate the:

- Overhead of FFT, ARIMA, and RRCF with each application and dataset.
- Performance of each algorithm using our proposed metric

TABLE I: Evaluation Metrics

Evaluation Metrics		
Metric	Meaning	Equation
Precision	Fraction of correctly labeled anomalies	$\frac{TruePositive}{TruePositive+FalsePositive}$
Recall	Fraction of anomalies reported compared to total injected anomalies	$\frac{TruePositive}{TruePositive+FalseNegative}$
Accuracy	Fraction of true anomalies from true predictions	$\frac{TruePositive}{TruePositive+TrueNegative}$
F1 Score	Harmonic mean of precision and recall	$\frac{TruePositive}{TruePositive+0.5(FalsePositive+FalseNegative)}$
Overhead	The log of seconds spent in an anomaly detection algorithm	

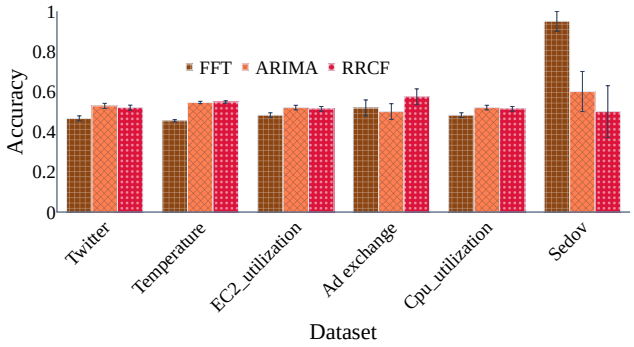


Fig. 3: Comparison of Accuracy

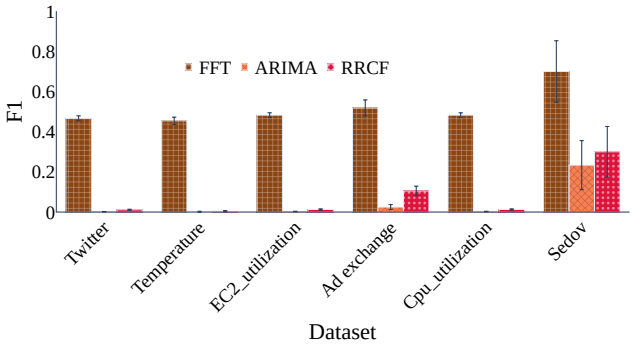


Fig. 4: Comparison of F1 Score

$\frac{accuracy \times F1\_score}{overhead}$.

- Robustness of each algorithm in terms of performance and overhead.

A. Experiment with benchmark datasets

In this experiment, we evaluate the performance of FFT, ARIMA, and RRCF using 6 benchmark datasets of real-time series data.

Figure 3 shows the accuracy of the algorithms FFT, ARIMA, and RRCF with the benchmark datasets. We see algorithms perform similar with the NAB real-time datasets, except Sedov, where FFT performs better than ARIMA and RRCF. Evaluating accuracy alone isn't sufficient to select a real-time anomaly detection algorithm, the rate of false positive and false negative predictions also plays a crucial role. Thus, the F1 score, being a combination of these, is a useful metric to consider. Figure 4 shows the calculated F1 score of each algorithm in this experiment. Viewing the F1 score in the figure, we can clearly see FFT outperforms ARIMA and RRCF. Aside from the Sedov dataset, ARIMA and RRCF suffer either poor recall or poor precision resulting in a negligible F1 score. Across algorithms for each dataset, we note similar error margins of F1 scores.

We combine the F1 score and the accuracy to obtain the "area", where $area = F1 \times accuracy$. We determine which algorithm will give us the best trade-off between the area and overhead by dividing the area by the overhead of each algorithm. Figure 5 depicts the ratio of area to overhead of each algorithm with the benchmark datasets. As the datasets have already been collected, overheads are smaller. In spite of this, ARIMA and RRCF have a higher overhead in comparison to FFT because of the complexity of data structures used. Both the higher overhead and lower F1 score contributes to the low $area/overhead$ score of ARIMA and RRCF.

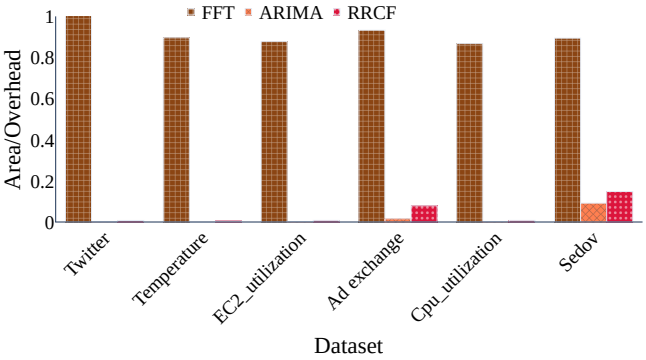


Fig. 5: Comparison of Area/Overhead

Table II provides a summary of the best performing algorithm for each metric with each of the datasets used.

B. Experiment with CHIMBUKO

In this experiment, we define α as the granularity of errors, and σ, as the sensitivity of each algorithm, and we randomly insert anomalous data across the 1000 timesteps of the application's run. While varying α and σ, we measure the accuracy of each algorithm with real-time CHIMBUKO data. In Figure 6, the y-axis shows the average accuracy of each algorithm, each bar

TABLE II: Summary Evaluation

Metrics			
Data set	Metric	Best Performing Algorithm	Value
Twitter	Accuracy	ARIMA	0.53 ± 0.013
	F1	FFT	0.47±0.013
	Overhead	FFT	0.43 ± 1.3
Temperature	Accuracy	RRCF	0.55 ± 0.005
	F1	FFT	0.45 ± 0.018
	Overhead	FFT	0.5 ± 1.28
EC2˙utilization	Accuracy	ARIMA	0.52 ± 0.012
	F1	FFT	0.482 ± 0.012
	Overhead	FFT	0.53 ± 1.5
Ad Exchange	Accuracy	ARIMA	0.57 ± 0.013
	F1	FFT	0.51 ± 0.039
	Overhead	FFT	0.53 ± 0.96
CPU Utilization	Accuracy	ARIMA	0.52 ± 0.01
	F1	FFT	0.48 ± 0.01
	Overhead	FFT	0.5 ± 1.2
Sedov	Accuracy	FFT	0.95 ± 0.05
	F1	FFT	0.7 ± 0.15
	Overhead	FFT	0.77 ± 0.5

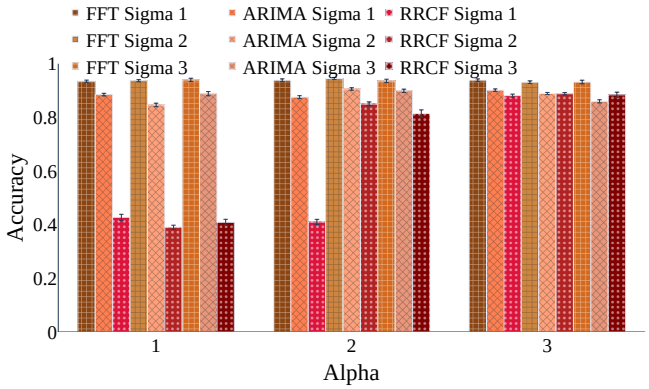


Fig. 6: Accuracy for $\sigma = 1 \rightarrow 3$, $\alpha = 1 \rightarrow 3$

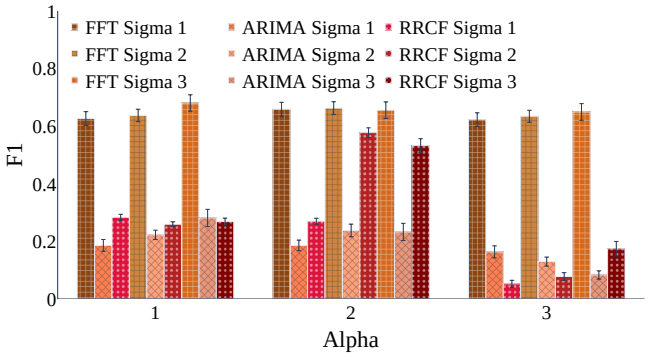


Fig. 7: F1 Score for $\sigma = 1 \rightarrow 3$, $\alpha = 1 \rightarrow 3$

represents an algorithm and σ value, and bars are grouped with the value of α increasing along the x-axis.

In figure 6, we note RRCF has a very low accuracy for detection of errors with small granularity, when $\alpha = 1$, where FFT and ARIMA both have an accuracy greater than 0.8. We also note that for every α FFT and ARIMA have a similar accuracy. Figure 7 shows the corresponding F1 scores for this experiment, here we see that, as compared with the benchmark datasets experiment, all algorithms have an increased F1 score, though FFT still clearly outperforms both ARIMA and RRCF.

Moreover, in all experiments, we note that the overhead of FFT is significantly lower than that of ARIMA and RRCF. Figure 8 shows the overhead during the first 100 timesteps for each algorithm with $\sigma = 1$ and α varied between 1 and 3. The timestep is represented along the x-axis and the y-axis represents the overhead as the log value of the time taken by each respective algorithm. The value of α is shown in the legend, preceded by the name of the algorithm. Initially, for a small number of timesteps, FFT has a higher overhead due to setup costs as it begins. The overhead of FFT quickly drops

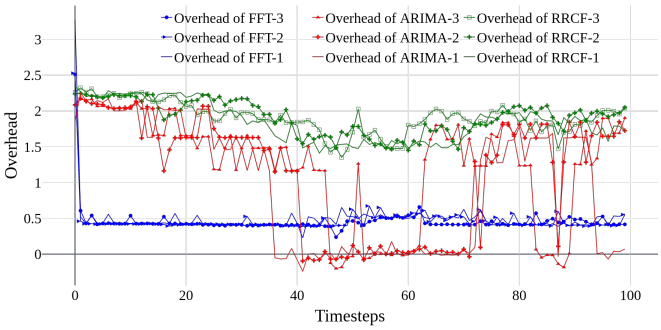


Fig. 8: Overhead for $\sigma = 1$, $\alpha = 1 \rightarrow 3$

and becomes stable, with an overall overhead lower than that of ARIMA and RRCF.

Figure 9 shows the $area/overhead$ of each algorithm as we vary both σ and α. It is important to note that the $area/overhead$ of FFT is greater than one due to an average overhead of less than 1. We see that as α increases, algorithms perform better as the granularity of the errors grows which

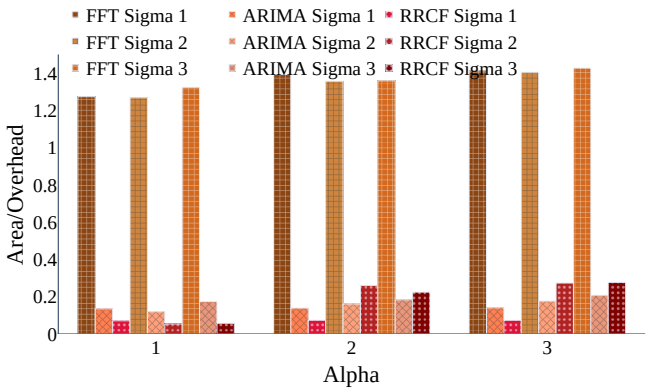


Fig. 9: Area /Overhead for Sigma, $\sigma = 1 \rightarrow 3$, Alpha,$\alpha = 1 \rightarrow 3$

leads to a decrease in the numbers of errors. FFT performs consistently with a higher F1 score, accuracy, lower overhead, and clearly outperforms ARIMA and RRCF when observing $area/overhead$. The $area/overhead$ of ARIMA and RRCF are similar, though ARIMA does show a small improvement as α increases.

The difference between the $area/overhead$ of FFT compared with that of ARIMA and RRCF is ≥ 0.9 in all cases, thus we conclude that FFT outperforms both ARIMA and RRCF and propose FFT as the algorithm to implement in CHIMBUKO.

VII. Related Work

As unsupervised ML methods are able to continuously process data, make decisions, and adapt to changing statistics, they are a good fit for real-time anomaly detection in streaming data [13].

Many algorithms chosen to detect anomalies in streaming data employ simple and lightweight statistical techniques [3] including hypothesis testing [20], wavelet analysis [15], Singular Value Decomposition (SVD) [16], ARIMA [28], and FFT [23].

FFT [23] is a statistical method which performs well in real time anomaly detection and is frequently used in the domain of visual computing [10]. Rasheed et al. [18] used FFT to perform outlier detection in spatial data by discovering regions of high frequency change. Hansheng et al. [19] leveraged the Spectral Residual (SR) model from the visual computing domain for to detect anomalies in real time series data. However, they also incorporate CNN in their approach which is not lightweight with 3 modules. Moreover, they focused their efforts on F1 score and accuracy at the expense of overhead. Lili et al. [27] used fractional FFT with the RX algorithm to detect anomalies in hyperspectral images with multi-pixel objects, but this approach is limited to the visual computing domain and not generalizable to real time data.

The use of FFT in many recent contributions in anomaly detection can be attributed to both its relatively simple nature and applicability to time series data. IBM's Maximo Asset Management [2], uses a fast fourier transform based anomaly detector. They treat data from devices as a signal and use FFT for pattern extraction, detecting anomalies in the signal according to a scoring function.

Anomaly detection algorithms have been proposed to reduce or remove negative effects caused by assumptions about the statistics of data. ARIMA [3] detects anomalies in data using temporal information. Bianco et al. have used ARIMA successfully to model seasonal time series data [4]. Several works have used ARIMA to predict anomalies in network traffic [26], [25], [17]. RRCF [9] uses a random data structure to detect anomalies in real time dynamic streaming data. Recent works [24], [7], have used RRCF to detect anomalies in streaming and time series data.

An algorithm's robustness, speed, and ability to scale are critical design considerations in real-time anomaly detection [21]. In ARIMA, detection of seasonal patterns is time consuming [21]. CHIMBUKO [12] provides speed and an ability to scale by considering only execution time in detection of performance anomalies while using HBOS for real-time analysis. This work extends anomaly detection in CHIMBUKO by additionally considering hardware performance counters without significantly increasing computational or memory overhead.

VIII. Conclusions

In this paper, we proposed a lightweight anomaly detection technique based on FFT for real-time data from production applications like CHIMBUKO. We compare its performance with two different anomaly detection techniques from machine learning and statistics domains on benchmark real-time datasets and real-time data from the production application CHIMBUKO. We also used a new metric to compare the performance of anomaly detection with real-time production applications to take into account the trade-off between accuracy, F1 score, and overhead. FFT consistently performs better than both ARIMA and RRCF and incurs negligible overhead. We limited our comparisons to these 3 algorithms as other algorithms will have a larger overhead rendering them unsuitable for real-time data analysis. Our goal was to find the anomaly detection algorithm that balances accuracy and overhead for real-time systems, and experiments validated that FFT is a good fit for lightweight real-time anomaly detection.

IX. Acknowledgement

The work was supported in part by the grant DE-SC0022843 funded by the U.S. Department of Energy, Office of Science. The work was also supported in part by the Exascale Computing Project (ECP) funded by the U.S. Department of Energy, Office of Science.

References

[1] Tau doccumentation. 37:n/a–n/a.

[2] Ibm maximo asset performance management saas-1˙cloud. 37:n/a–n/a, 12 2021.

[3] Subutai Ahmad, Alexander Lavin, Scott Purdy, and Zuha Agha. Unsupervised real-time anomaly detection for streaming data. Neurocomputing, 262:134–147, 2017. Online Real-Time Learning Strategies for Data Streams.

[4] A.M. Bianco, M. García Ben, E.J. Martínez, and V.J. Yohai. Outlier detection in regression models with arima errors using robust estimates. Journal of Forecasting, 20(8):565 – 579, 2001. Cited by: 81.

[5] C. Chatfield and D. L. Prothero. Box-jenkins seasonal forecasting: Problems in a case-study. Journal of the Royal Statistical Society: Series A (General), 136(3):295–315, 1973.

[6] Ewa Deelman, Tom Peterka, Ilkay Altintas, Christopher D Carothers, Kerstin Kleese van Dam, Kenneth Moreland, Manish Parashar, Lavanya Ramakrishnan, Michela Taufer, and Jeffrey Vetter. The future of scientific workflows. The International Journal of High Performance Computing Applications, 32(1):159–175, 2018.

[7] Minjung Gim, Jin-Hwan Cho, and Dong Heon Cheo. Anomaly detection in sensing data based on rrcf.

[8] Markus Goldstein and Andreas Dengel. Histogram-based outlier score (hbos): A fast unsupervised anomaly detection algorithm. KI-2012: poster and demo track, 9, 2012.

[9] Sudipto Guha, Nina Mishra, Gourav Roy, and Okke Schrijvers. Robust random cut forest based anomaly detection on streams. In Maria Florina Balcan and Kilian Q. Weinberger, editors, Proceedings of The 33rd International Conference on Machine Learning, volume 48 of Proceedings of Machine Learning Research, pages 2712–2721, New York, New York, USA, 20–22 Jun 2016. PMLR.

[10] Xiaodi Hou and Liqing Zhang. Saliency detection: A spectral residual approach. In In Proc. IEEE Conference on Computer Vision and Pattern Recognition CVPR '07, pages 1–8, 2007.

[11] Christopher Kelly, Sungsoo Ha, Kevin Huck, Hubertus Van Dam, Line Pouchard, Gyorgy Matyasfalvi, Li Tang, Nicholas D'Imperio, Wei Xu, Shinjae Yoo, et al. Chimbuko: A workflow-level scalable performance trace analysis tool. In ISAV'20 In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization, pages 15–19. 2020.

[12] Christopher Kelly, Sungsoo Ha, Kevin Huck, Hubertus Van Dam, Line Pouchard, Gyorgy Matyasfalvi, Li Tang, Nicholas D'Imperio, Wei Xu, Shinjae Yoo, and Kerstin Kleese Van Dam. Chimbuko: A workflow-level scalable performance trace analysis tool. In ISAV'20 In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization, ISAV'20, page 15–19, New York, NY, USA, 2020. Association for Computing Machinery.

[13] Alexander Lavin and Subutai Ahmad. Evaluating real-time anomaly detection algorithms – the numenta anomaly benchmark. In 2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA), pages 38–44, 2015.

[14] Alexander Lavin and Subutai Ahmad. "evaluating real-time anomaly detection algorithms – the numenta anomaly benchmark", fourteenth international conference on machine learning and applications. 37:n/a–n/a, 12 2015.

[15] Wei Lu and Ali A. Ghorbani. Network anomaly detection based on wavelet analysis. EURASIP J. Adv. Signal Process, 2009, jan 2009.

[16] Ajay Mahimkar, Zihui Ge, Jia Wang, Jennifer Yates, Yin Zhang, Joanne Emmons, Brian Huntley, and Mark Stockert. Rapid detection of maintenance induced changes in service performance. In Kenjiro Cho and Mark Crovella, editors, Proceedings of the 2011 Conference on Emerging Networking Experiments and Technologies, Co-NEXT '11, Tokyo, Japan, December 6-9, 2011, page 13. ACM, 2011.

[17] Eduardo H. M. Pena, Marcos V. O. de Assis, and Mario Lemes Proença. Anomaly detection using forecasting methods arima and hwds. In 2013 32nd International Conference of the Chilean Computer Science Society (SCCC), pages 63–66, 2013.

[18] Faraz Rasheed, Peter Peng, Reda Alhajj, and Jon Rokne. Fourier transform based spatial outlier mining. In Emilio Corchado and Hujun Yin, editors, Intelligent Data Engineering and Automated Learning - IDEAL 2009, pages 317–324, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[19] Hansheng Ren, Bixiong Xu, Yujing Wang, Chao Yi, Congrui Huang, Xiaoyu Kou, Tony Xing, Mao Yang, Jie Tong, and Qi Zhang. Time-series anomaly detection service at microsoft. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery Data Mining, KDD '19, page 3009–3017, New York, NY, USA, 2019. Association for Computing Machinery.

[20] Bernard Rosner. Percentage points for a generalized esd many-outlier procedure. Technometrics, 25(2):165–172, 1983.

[21] Meir Toledano, Ira Cohen, Yonatan Ben-Simhon, and Inbal Tadeski. Real-time anomaly detection system for time series at scale. In Archana Anandakrishnan, Senthil Kumar, Alexander Statnikov, Tanveer Faruquie, and Di Xu, editors, Proceedings of the KDD 2017: Workshop on Anomaly Detection in Finance, volume 71 of Proceedings of Machine Learning Research, pages 56–65. PMLR, 14 Aug 2018.

[22] Granville Tunnicliffe Wilson. Time series analysis: Forecasting and control,5th edition, by george e. p. box, gwilym m. jenkins, gregory c. reinsel and greta m. ljung, 2015. published by john wiley and sons inc., hoboken, new jersey, pp. 712. isbn: 978-1-118-67502-1. Journal of Time Series Analysis, 37:n/a–n/a, 03 2016.

[23] Charles Van Loan. Computational Frameworks for the Fast Fourier Transform. Society for Industrial and Applied Mathematics, 1992.

[24] Qifan Wang, Bo Yan, Hongyi Su, and Hong Zheng. Anomaly detection for time series data stream. In 2021 IEEE 6th International Conference on Big Data Analytics (ICBDA), pages 118–122, 2021.

[25] Asrul H. Yaacob, Ian K.T. Tan, Su Fong Chien, and Hon Khi Tan. Arima based network anomaly detection. In 2010 Second International Conference on Communication Software and Networks, pages 205–209, 2010.

[26] H. Zare Moayedi and M.A. Masnadi-Shirazi. Arima model for network traffic prediction and anomaly detection. In 2008 International Symposium on Information Technology, volume 4, pages 1–6, 2008.

[27] Lili Zhang, Jiachen Ma, Baozhi Cheng, and Fang Lin. Fractional fourier transform-based tensor rx for hyperspectral anomaly detection. Remote Sensing, 14(3), 2022.

[28] Yin Zhang, Zihui Ge, Albert Greenberg, and Matthew Roughan. Network anomography. In Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement, IMC '05, page 30, USA, 2005. USENIX Association.