RAMP: Real-Time Anomaly Detection in Scientific Workflows

J. Dinal Herath*, Changxin Bai†, Guanhua Yan*, Ping Yang*, and Shiyong Lu†
* State University of New York at Binghamton, Binghamton, NY, USA
† Wayne State University, Detroit, MI, USA

***Abstract*—Research integrity is crucial to ensuring the trustworthiness of scientific discoveries. This work is aimed at detecting misbehaviors targeting scientific workflows, which are computing paradigms widely used to facilitate scientific collaborations across multiple geographically distributed research sites. We develop a new system called *RAMP* (Real-Time Aggregated Matrix Profile) for real-time anomaly detection in scientific workflow systems. RAMP builds upon an existing time series data analysis technique called Matrix Profile to detect anomalous distances among sub-sequences of event streams collected from scientific workflows in an online manner. Using an adaptive uncertainty function, the anomaly detection model is dynamically adjusted to prevent high false alarm rates. RAMP can incorporate user feedback on reported anomalies and modify model parameters to improve anomaly detection accuracy. Our experimental results from applying RAMP to the logs generated by DATAVIEW, a scientific workflow platform, show that RAMP is able to identify a varied range of anomalies with high accuracy for both interleaved and non-interleaved workflow executions in real time.**

I. Introduction

Research integrity is crucial to ensuring the trustworthiness of scientific discoveries. Scientific misconducts not only cause reputation damages to researchers and institutions involved in the scientific community, but also can have severe real-life implications if dubious research results are transitioned into practical use, such as medicine production and dietary guidelines. Although there have been various regulations to prevent or deter misconducts in scientific research, such misbehaviors are still not uncommon, according to a report in 2009 revealing that 2% of scientists surveyed had falsified, fabricated, or modified their research data [13].

It is thus important to enhance existing cyberinfrastructures used for scientific research activities with capabilities to detect misbehaviors at their early stages before they contaminate the eventual scientific discovery results. This work aims at detecting misbehaviors targeting scientific workflows, which are computing paradigms widely used to facilitate scientific collaborations across multiple geographically distributed research sites. Popular scientific workflows include e.g. Montage used by astronomists for image mosaics of the sky [10], CyberShake for generating seismic hazard maps [18], and the myExpriment social network site for bioinformatics researchers [15].

This work aims to develop new techniques that can detect anomalies in scientific workflows in *real time*. The term "real time" is similar to that in [3], [8], which means that the anomaly detection model must observe a data record in a sequential manner and any processing, learning, or anomaly identification must be done before the arrival of the next data record. Real-time anomaly detection has the advantage of catching perpetrators' misbehaviors at their early stages so that the altered or falsified data or code can be prevented from propagating into downstream scientific processes. Although practically appealing, real-time anomaly detection requires us to tackle the following technical challenges. Firstly, the anomaly detection algorithm must be efficiently implemented in order to keep up with the velocity of the event streams observable in scientific workflows. Secondly, the anomaly detection algorithm must be adaptive to situations where there exists only limited supervised information initially. Lastly, the anomaly detection algorithm should be able to adjust its parameters dynamically based on human users' feedback on its reported anomalies.

Against this backdrop, we develop a new system called *RAMP* (*Real-Time Aggregated Matrix Profile*) for real-time anomaly detection in scientific workflow systems. RAMP not only detects anomalies, but also provides insight into what features in a multidimensional time series may have caused it. RAMP builds upon an existing time series data analysis technique called Matrix Profile to detect anomalous distances among sub-sequences of event streams collected from scientific workflows in an online manner. Using an adaptive uncertainty function, the anomaly detection model is dynamically adjusted to facilitate fast convergence to a stable state. RAMP can also incorporate user feedback on reported anomalies and retrain model parameters to improve anomaly detection accuracy. We have implemented RAMP to parse logs generated by DATAVIEW, a scientific workflow platform built upon Amazon EC2 [17], and detect anomalous activities targeting DATAVIEW in an online manner.

In a nutshell, our main contributions are as follows:

- We tailor the vanilla Matrix Profile method, which operates on data in a batch wise manner, to real-time anomaly detection with two main modifications: using relative distances among sub-sequences to avoid inherent biases in Euclidean distance computation and constraining calculation of distance profiles with a small-sized training base to facilitate online model training.
- We introduce a new adaptive training mechanism to reduce false alarm rates commonly plaguing anomaly detection systems in practice. Our method uses a novel uncertainty function that models evolving beliefs in flagged anomalies over time to adjust model parameters with new

anomalies reported. This technique can prevent RAMP from triggering repetitive alarms due to the same type of anomalies occurred.
- We empower RAMP with an optional human-in-the-loop training scheme. To improve anomaly detection accuracy, our method carefully modifies RAMP's model parameters based on human users' feedback.
- We conduct intensive experiments to evaluate the effectiveness of RAMP. We compare the performance of RAMP against those of two state-of-the-art real-time anomaly detection models and our results show that RAMP has superior performances in various anomaly situations while achieving real-time responsiveness.

The remainder of the paper is organized as follows. Section II provides the background information of this work, including scientific workflows and the Matrix Profile method. Section III discusses the threat model as well as different types of anomalies we aim to detect in this work. Section IV and Section V present an overview of RAMP and its design details, respectively. Experimental results of RAMP are given in Section VI. We discuss related work in Section VII and draw concluding remarks in Section VIII.

II. Background

This section provides an overview of scientific workflows and the Matrix Profile, the base model used to build RAMP.

A. Scientific Workflows

Scientific workflow is a cyberinfrastructure paradigm for automating and accelerating data processing and data sharing in the scientific community. Figure 1 shows a diagnosis recommendation workflow [4]. This workflow consists of five workflow tasks $T_1 - T_5$, each of which represents a computational or analytical step in the workflow. Task T_1 extracts 27 selected features and diagnosis labels from the unstructured textual medical datasets. The raw datasets are then split into training datasets and testing datasets (task T_2). Task T_3 labels the unlabeled training data using the frequent 1-itemset value calculated from all labeled training data. Task T_4 predicts the diagnosis label for the data in the testing datasets. Task T_5 outputs the recommended diagnosis labels.

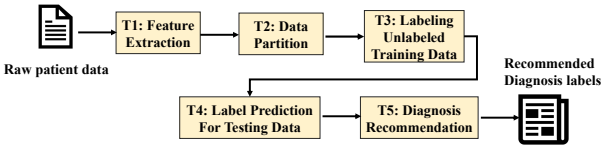


Fig. 1: A Diagnosis Recommendation Workflow

B. DATAVIEW and DATAVIEW Logs

DATAVIEW is a big data workflow management system, which uses Amazon EC2 as the public cloud computing environment. DATAVIEW logs record the real-time status of scientific workflows executed on EC2. For each workflow, DATAVIEW creates textual log entries specifying the task execution status (e.g. task-start, task-completion), the communication between the local machine and the EC2 VMs (e.g. task-send, task-receive), and the machine provisioning status (e.g. machine-idle, machine-ready), which are called *events*. Each log entry is associated with a timestamp, which specifies the date and time of an event. The log also contains the IP address of VMs on which workflow tasks are executed.

C. Matrix Profile

Matrix Profile [28] is an incrementally updatable model that enables the identification of similar (called *motifs*) and dissimilar (called *discords*) patterns in a given time series. When new data is appended to the original time series input, Matrix Profile does not need to re-compute motifs and discords from the beginning using all the data points, instead it is able to change the existing results with little computational overhead. This incremental updatable nature enables Matrix Profile to identify previously un-identified motifs that appear due to a new data stream with relative ease. Matrix Profile's incremental updatability, fast execution, and low need for parametric tuning (*i.e. only one parameter to tune*) ([5], [27], [28], [30]) makes it an attractive model for real time machine learning applications.

We begin by first defining a *univariate time series* T which is a sequence of real numbers $T = t_1, t_2, \ldots, t_n$. The Matrix Profile gives insight about the global similarity or dissimilarity in a time series, but this is computed with respect to local *sub-sequences*. A *sub-sequence* $T_{i,m}$ of T is a continuous subset of the values from T with a given length m starting at position i (i.e. $T_{i,m} = t_i, t_{i+1}, \ldots, t_{i+m-1}$ where $1 \leq i \leq n - m + 1$). For any given sub-sequence in a time series, it is possible to compute the Euclidean distance from itself to all other sub-sequences. An ordered vector of the Euclidean distances between a given sub-sequence $T_{i,m}$ and an set of all sub-sequences $[T_{1,m}, T_{2,m}, ..., T_{n-m+1,m}]$ is called a *distance profile* D. By extension, a multivariate time series $\mathbf{T}$ of d dimensions is a set of co-evolving univariate time series where $\mathbf{T} = [T^{(1)}, T^{(2)}, ..., T^{(d)}]$ and a multivariate subsequence is given by $\mathbf{T_{i,m}} = [T^{(1)}_{i,m}, T^{(2)}_{i,m}, ..., T^{(d)}_{i,m}]$.

Let T be a complete univariate time series and m be the length of the sub-sequence. Matrix Profile works as follows. First, it computes the distance profile D_i for every sub-sequence $T_{i,m}$. The Matrix profile value at step i is obtained as the minimum recorded value in D_i, excluding the Euclidean distance from $T_{i,m}$ to itself which is trivially 0. Repeating this process for the complete time series results in an ordered vector of minimum Euclidean distance values corresponding to each sub-sequence, which is called a *Matrix Profile*. A small value in the Matrix Profile indicates that the sub-sequence pattern is observed elsewhere in the time series (i.e. a *motiff*). An abnormally large Matrix Profile value indicates that the corresponding sub-sequence is not observed elsewhere in the time series and hence may be a *discord*.

III. Threat Model and Attacks

We assume that the workflow code does not change often and that the workflow logs are protected and are not tampered.

We also assume that an attacker may exploit the vulnerabilities in the workflow to modify the workflow tasks and/or the communication between two tasks to alter scientific results. In addition, an attacker can attack the workflow systems using the Denial Of Services(DOS) attack.

We classify anomaly situations into the following two categories: **(1) Level-1 anomalies**: anomalies resulted from direct attacks or malfunctions of DATAVIEW; **(2) Level-2 anomalies**: attacks that attempt to hide the true intent of an attack and confound the anomaly detection model.

A. Level-1 Anomalies

We consider the following four Level-1 anomaly instances.

L1A1-Unexpected scheduler change: DATAVIEW provides several task scheduling options, which deploy the workflow tasks on different numbers of VMs. We consider the situation where tasks are scheduled and executed on the VMs in a pattern that is not predefined. This can be caused by an attacker who intends to increase the load on DATAVIEW or as a part of an adversarial attack (L2A1) described below.

L1A2-DOS attack: In DATAVIEW, after EC2 VMs are provisioned, there are continuous communications between the local machine used by a user and the VMs on which the workflow tasks are executing, including task specifications, task execution status, and VM status. We consider the DOS attacks or spikes in network traffic which may slow down the execution of scientific workflows.

L1A3-Task manipulation: This attack is performed by malicious users who have access to the source code of the workflow tasks. The malicious users modify the task code or inject pre-computed values into the task execution to manipulate the workflow result. RAMP detects L1A3 attacks that result in an increase/decrease on the task execution time.

L1A4-Workflow structure manipulation: When the task code is unavailable, a malicious user injects a task or change the workflow structure to manipulate the workflow result.

B. Level-2 Anomalies

We present two adversarial attacks designed to confound an anomaly detection model.

L2A1-Workflow structure manipulation with Scheduler change: The attacker performs a scheduler change attack (L1A1) and a workflow structural change attack (L1A4) simultaneously. In this attack, the attacker aims to use the VM provisioning change to mask the code change in the task, which modifies the final results of the scientific workflow.

L2A2-Task manipulation with DOS attack: This combined attack performs a DOS attack (L1A2) and a task manipulation attack (L1A3) simultaneously. Both attacks affect the execution time of workflows. A combined attack of this nature aims to mask task manipulation occurring during a DOS attack.

IV. Overview of RAMP

This section provides an overview of RAMP, which detects anomalies in scientific workflows based on the logs generated by DATAVIEW. Unlike the vanilla Matrix Profile which identifies both $motif$ and $discords$, RAMP is specifically designed to capture only $discords$ (or anomalies) in a time series.

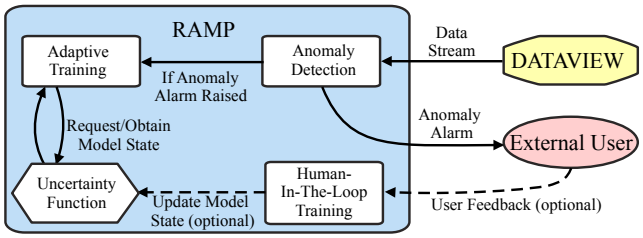


Fig. 2: The Architecture of RAMP

log_1	2019/06/13-12:03:01.340 machine-ready 204.236.200.9
log_2	2019/06/13-12:03:01.763 machine-ready 54.196.14.157
log_3	2019/06/13-12:03:02.184 machine-ready 54.147.255.97
log_4	2019/06/13-12:03:02.601 machine-ready 34.204.71.118

TABLE I: Log entries for VM Provisioning in DATAVIEW

Figure 2 gives the architecture of RAMP. The DATAVIEW log entries are parsed into a real-time time series with three dimensions (i.e. three features). The first is the time difference between two successive log entries computed in millisecond precision. The second dimension reflects the change in the task/process executed between two consecutive logs. In Table I, the term *machine-ready* reflects the machine provisioning occurring at each log step. Therefore, the time series would reflect the same task [machine-ready→machine-ready] occurring for three turns. The final dimension would reflect the change in IP address between two successive log entries as [[204.236.200.9 $\rightarrow$ 54.196.14.157] , [54.196.14.157 $\rightarrow$ 54.147.255.97] , [54.147.255.97 $\rightarrow$ 34.204.71.118]]. Once the log entries are parsed, sub-sequences are formed and given to RAMP. Considering a sub-sequence length of 3, the first sub-sequence would be concatenated values of time series entries from $[log_1, log_2, log_3]$ followed by the second sub-sequence $[log_2, log_3, log_4]$.

RAMP has three main components, namely an *Anomaly Detection* module, an *Adaptive Training* module, and an optional *Human-in-the-Loop training* module. The *Anomaly Detection* module detects anomalies based on a modified Matrix Profile model. At each time step, the *Anomaly Detection* module takes as input a sub-sequence of data stream and computes a weighted aggregated anomaly score, which signifies the possibility that the input sub-sequence is an anomaly or not. The anomaly score aggregates the prediction results of individual Matrix Profile models operated on different dimensions of the input sub-sequence. The *Anomaly Detection* module is also capable of providing insight into the individual features that result in the anomaly.

As it is unclear what types of attacks or anomalies could happen in the future, RAMP utilizes a semi-supervised model where RAMP learns the correct behaviour of workflow execution in the first few workflow runs and then identifies instances that heavily deviate from it as potential anomalies. This approach is used in other real-time machine learning models (e.g. anomaly detection models in Numenta Anomaly

Benchmark (NAB) [1]), where an initial grace period is defined such that models can learn the correct behaviour of real time data. This grace period, however, may not be sufficient for models to continuously update its internal state and detect anomalies with high accuracy. The *Adaptive Training* module is designed to facilitate fast state convergence in real time.

The *Adaptive Training* module is invoked to update model weights whenever the anomaly detection module flags an anomaly. These weights are updated according to the anomaly score reported by the *Anomaly Detection* module and an *uncertainty function*, which probabilistically captures the model state. If its probabilistic value is close to 1, RAMP believes that the anomaly flagged is a true positive with high certainty, whereas the opposite is considered if this value is close to 0. The *uncertainty function* assumes that anomalies are less likely to occur in the first few executions of a scientific workflow and the likelihood becomes higher with more runs. The intuition behind this assumption is that an attacker has too little information to attack a scientific workflow in its first few execution runs but the situation changes once he gains more knowledge about the workflow. The *Adaptive Training* module ensures that RAMP will continuously learn the temporal behaviour of a time series and subsequently improves the overall performance.

Finally, RAMP uses an optional *human-in-the-loop training* module to improve anomaly detection accuracy based on human feedback. More specifically, given the true positives identified by human users, this module enforces the corresponding model weights such that similar anomalies encountered in the future are more likely to be caught by RAMP.

V. Algorithm Details

This section presents the details of the three RAMP modules: anomaly detection, adaptive training, and human-in-the-loop training. RAMP runs in a time-stepped manner where each time step corresponds to a new event from the input stream. At each time step the anomaly detection module is invoked and if an anomaly is flagged, the adaptive training is called to update model parameters. For every M user defined time steps, RAMP invokes the human-in-the-loop training to process human feedback, if there is any.

A. Anomaly Detection Module

RAMP uses a modified version of Matrix Profile for real-time anomaly detection. Our description here uses the same notations as the original Matrix Profile discussed in Section II-C. In the following, we first explain our key modifications to Matrix Profile and then present the details of our anomaly detection algorithm.

1) Modifications to Matrix Profile:: One modification we made to the Matrix Profile is limiting the number of subsequences compared. For a given sub-sequence, Matrix Profile computes the Euclidean distance with respect to all other subsequences and identifies the minimum distance. Therefore, a repeated anomaly instance would cause false negatives due to the previous anomaly instance being part of the all subsequence set. RAMP, in contrast, uses a semi-supervised model where only the first $M - m + 1$ sub-sequences (i.e. sub-sequences up to the M-th item in the time series) are considered as references for comparison with no anomalies. We define the *training base* as $\mathbf{T}^{'} = [\mathbf{T_{1,m}}, \mathbf{T_{2,m}}, ..., \mathbf{T_{M\text{-}m+1,m}}]$ with d dimensions (i.e. $\mathbf{T_{1,m}} = [T^{(1)}_{1,m}, T^{(2)}_{1,m}, ..., T^{(d)}_{1,m}]$). We illustrate in Figure 3 the demarcation of sub-sequences and the initial training base for a univariate time series ($d = 1$) where $m = 3$ and $M = 50$.

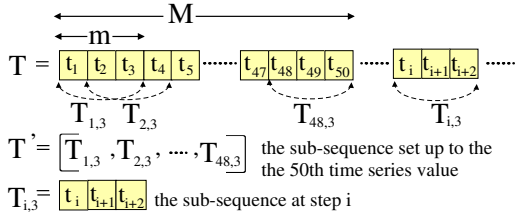


Fig. 3: Demarcation of sub-sequences

Our second modification is that, instead of computing the absolute Euclidean distance, we compute relative distances among sub-sequences. The purpose behind this modification is to overcome the inherent bias of Euclidean distance towards numerically larger data points. Given two univariate sub-sequences $T_{1,m} = [t_1, t_2, ..., t_m]$ (an input sub-sequence) and $T^{'}_{1,m} = [t^{'}_1, t^{'}_2, ..., t^{'}_m]$ (a sub-sequence used for comparison), the relative distance is computed by $\frac{\sum_{l=1}^{m} |t_l - t^{'}_l|}{\sum_{l=1}^{m} |t^{'}_l|}$.

2) Algorithm description: Our anomaly detection algorithm is given in Algorithm 1. For every time step i, *Anomaly-Detection* computes an aggregated anomaly score, β_i, which indicates the likelyhood of an anomaly. In order to compute β_i, for each dimension of the input subsequence $\mathbf{T_{i,m}}$, we first calculate the minimum relative distance between $\mathbf{T_{i,m}}$ and any subsequence in the training base $\mathbf{T}^{'}$. The aggregated anomaly score β_i is updated to be the sum of the minimum relative distances over all d dimensions (Line 2-12). The individual contribution of each dimension towards the aggregated anomaly score, which is calculated as the ratio of the minimum relative distance between $\mathbf{T_{i,m}}$ and any subsequence in the training base $\mathbf{T}^{'}$ to β_i, is saved into a contribution list $C_i = [C_i^{(1)}, C_i^{(2)}, ..., C_i^{(d)}]$ (Line 13-14). For example, $C_i^{(1)} = 1$ indicates that the anomaly score is fully decided by the first dimension in the time series.

It is noted that the aggregated anomaly score β_i is affected by a *unique* combination of d subsequences in the training base $\mathbf{T}^{'}$, each having a minimum relative distance from the current input subsequence $\mathbf{T_{i,m}}$ at one of the d dimensions. As there are $M - m + 1$ subsequences in the training base, there are $(M - m + 1)^d$ possible combinations over d dimensions. Our algorithm keeps a weight for each of these combinations, reflecting the model's confidence level in the aggregated anomaly score if it is derived from this combination. In Algorithm 1, the index of the unique combination responsible for the computation of β_i is stored in key (Line 10). The eventual anomaly score β_i is derived by multiplying it by the weight indexed by key (Line 15-16). It is easy to see

Procedure `AnomalyDetection`

Data: the time step i, the input sub-sequence $\mathbf{T_{i,m}} = [T_{i,m}^{(1)}, T_{i,m}^{(2)}, ..., T_{i,m}^{(d)}]$ at time step i, the length of the sub-sequence m, the window size for periodic user feedback M, the sample sub-sequence for comparison $\mathbf{T}^{'}$, the records to be updated for training R and H, the number of dimensions d, the user defined threshold θ, and the weights W.

1 $\beta_i = 0$, $C_i = \emptyset$, $key = 0$, $D_{min} = \emptyset$
2 **for** $j = 1$ *to* d **do**
3 $\quad min\_rd = +\infty$, $min\_k = -1$
4 $\quad$ **for** $k = 1$ *to* $M - m + 1$ **do**
5 $\quad\quad relativeDistance = \frac{\sum_{l=1}^{m} |T_{k,m}^{(j)}[l] - T_{i,m}^{(j)}[l]|}{\sum_{l=1}^{m} |T_{k,m}^{(j)}[l]|}$
6 $\quad\quad$ **if** $relativeDistance > min\_rd$ **then**
7 $\quad\quad\quad min\_rd = relativeDistance$
8 $\quad\quad\quad min\_k = k$
$\quad$ **end**
9 $\quad R[j, i\%M + 1] = min\_k$
10 $\quad key \mathrel{+}= (M - m + 1)^{j-1} \cdot (min\_k - 1)$
11 $\quad \beta_i \mathrel{+}= min\_rd$
12 $\quad D_{min}[j] = min\_rd$
end
13 **for** $j = 1$ *to* d **do**
14 $\quad C_i[j] = D_{min}[j]/\beta_i$
end
15 **if** key *exists in* W **then**
16 $\quad \beta_i = W[key] \times \beta_i$
17 $H[i\%M + 1] = \beta_i/\theta$
18 **if** $\beta_i > \theta$ **then**
$\quad AnomalyDetected = True$
else
$\quad AnomalyDetected = False$
end
19 **return** $[AnomalyDetected, C_i, R, H]$

Algorithm 1: *Anomaly Detection* procedure

a challenge due to the curse of dimentionality: if d is large, it is expensive to store all $(M - m + 1)^d$ possible weights. To circumvent this problem, we use a hash table W to store only updated weights, while assuming that those weights not in W take a default value of 1. Our experimental results in Section VI-C show that only a small fraction of possible weights are updated by RAMP in practice.

Recall that the human-in-the-loop training module is called by RAMP every M time steps to process any feedback from human users. As human users may overrule the anomaly detection results by RAMP in the past M time steps, RAMP should remember the indices of those weights that have been used to compute the anomaly scores during this period in case that they need to be revised based on the user feedback. To this end, we define the following two auxiliary data structures. Matrix R of dimensions $d \times M$ records the indices of weights that have been updated in the past M time steps. More specifically, entry $R[j, k]$ stores the index of the subsequence within the training base $\mathbf{T}^{'}$ that has the minimum relative distance for dimension $j \in [1, d]$ at the k-th time step among the past M ones (i.e., $k \in [1, M]$). This is done by Line 9 in Algorithm 1. Additionally, array H of length M stores the ratios of the derived anomaly scores to the user-defined threshold (i.e., $\frac{\beta_i}{\theta}$) for the past M time steps (Line 17).

The eventual aggregated anomaly score β_i is compared against a user-defined threshold θ (Line 18): if $\beta_i > \theta$, an anomaly is flagged, which further triggers the execution of the adaptive learning module; otherwise, the anomaly detection module terminates. The time complexity of the anomaly detection module is bounded by $\mathcal{O}(m(M - m + 1)d)$.

B. Adaptive Training

The original Matrix Profile model does not adjust its model parameters based on the anomalies it has detected, which can cause repetitive false alarms for a long period of time. To reduce false positive rate, the adaptive training module of RAMP, which is called when an alarm is raised by the anomaly detection module, adjusts not only the model parameters affected at the current time step but also those that may be affected in the near future.

Procedure `AdaptiveTraining`

Data: the time step i, the uncertainty value p_i at step i, the sub-sequence length m, the window size for periodic user feedback M, the user defined threshold θ, the records R and H used for training, the number of dimensions d, the training bias values α, and the weights W.

1 $keys = zeros[1, 2m + 1]$
2 **for** $k = 1$ *to* $2m + 1$ **do**
3 $\quad$ **for** $j = 1$ *to* d **do**
4 $\quad\quad keys[k] \mathrel{+}= (M-m+1)^{j-1} \cdot (R[j, i\%M+1] - m + k - 1)$
$\quad$ **end**
5 $\quad$ **if** $keys[k]$ *does not exist in* W **then**
6 $\quad\quad W[keys[k]] = 1$
7 $\quad$ **if** $k == m + 1$ **then**
8 $\quad\quad \beta_i^{unweighted} = \frac{H[i\%M+1] \cdot \theta}{W[keys[k]]}$
9 $\quad\quad W[keys[k]] = \frac{W[keys[k]]}{2H[i\%M+1]} \cdot \alpha[k] \cdot (1 - p_i)$
10 $\quad\quad H[i\%M + 1] = \frac{W[keys[k]] \cdot \beta_i^{unweighted}}{\theta}$
$\quad$ **else**
11 $\quad\quad W[keys[k]] = \frac{W[keys[k]]}{2H[i\%M+1]} \cdot \alpha[k] \cdot (1 - p_i)$
$\quad$ **end**
end
12 **return** $[W, H]$

Algorithm 2: *Adaptive Training* procedure

Algorithm 2 gives the *Adaptive Training* procedure. Recall that there are $(M - m + 1)^d$ possible weights affecting the anomaly scores and they are indexed by the keys to hash table W that stores updated weights. To achieve real time training, we use a training heuristic, which is experimentally validated in Section VI-D, to select only $(2m + 1)$ weight values for updating irrespective of the dimensionality d of the input. The middle $m + 1$ weight values, which correspond to $k = m + 1$ in Algorithm 2, are called the *anomaly-inducing weights*, as they are the same ones used to update the anomaly score by the anomaly detection module (see Line 16 in Algorithm 1). The keys indexing the $(2m + 1)$ weights in hash table W to be updated are calculated in Lines 3-4.

These selected weights are modified in Lines $7-11$ according to the following equation:

$$W[key] = \frac{W[key]}{2H[i\%M+1]} \cdot \alpha[k] \cdot (1-p_i), \qquad (1)$$

where $W[key]$ is the weight to be updated, $H[i\%M+1]$ is the ratio of the anomaly score computed ($\beta_{i\%M+1}$) to threshold θ, $\alpha[k]$ is a training bias value that captures temporal correlation for the k-th chosen weight, and p_i reflects the uncertainty level at time step i. Due to weight updating, $H[i\%M+1]$, which stores $\frac{\beta_i}{\theta}$, should also be updated. This is done by Line 10 in Algorithm 2 where the unweighted anomaly score $\beta_i^{unweighted}$ (i.e., the unweighted value before Line 16 of Algorithm 1) is first derived and then used to update $H[i\%M+1]$ along with the new weight.

The rationale behind Eq. (1) is that future anomaly score calculations using the same weight indexed by key should result in smaller values for β_i to avoid repetitive alarms. As the adaptive training module is called when an anomaly is detected (i.e., $\beta_i > \theta$), $H[i\%M+1]$, which is calculated as β_i/θ (see Line 17 in Algorithm 1) should always be greater than 1. The first portion of the formula $\frac{W[key]}{2H[i\%M+1]}$ aims to reduce the weight value so that a future use of the same weight on the same unweighted anomaly score will result in exactly half of the threshold θ. This is because, assuming that the unweighted anomaly score is β_0 and the old weight is w_0, we have: $H[i\%M=1] = w_0\beta_0/\theta$; as $\frac{W[key]}{2H[i\%M+1]} = \theta/(2\beta_0)$, using it on the same unweighted anomaly score β_0 leads to a weighted anomaly score of $\theta/2$.

Uncertainty function p_i***:*** Each weight in Eq. (1) is also multiplied by a factor of $1-p_i$, where $p_i \in [0,1]$ is an *uncertainty function* capturing the model state:

$$p_i = \begin{cases} 1-\exp((K_i)^b - (i)^b), & \text{if } i > M \\ 0, & \text{otherwise} \end{cases} \qquad (2)$$

Here, i is the current time step, K_i is a state variable updated upon user feedback (see Section V-C), and $b \in [0,1]$ is a user-defined bias parameter used to define the rate at which p_i converges to 1. When p_i is close to 0, RAMP believes that the reported anomalies are false positives with a high certainty. When p_i is close to 1, RAMP believes that the reported anomalies are likely to be true positives. At the beginning of RAMP, K_i is initialized to M whereas b is given by the user. The greater the b, the faster the convergence. RAMP uses the first M input values to build $\mathbf{T}^{'}$. Therefore, p_i is 0 when $i \leq M$ and starts increasing when $i = M$. Without any user feedback p_i will gradually increase to 1 with no sudden drops. This reflects our assumption that anomalies are more likely to occur with longer running time. In Section V-C, we illustrate how the uncertainty function changes with the user feedback.

Training bias α***:*** As the training procedure updates $2m+1$ temporally correlated weights, each one is multiplied by its respective training bias. The training bias α is a vector of size $2m+1$ that captures the temporal correlation among $2m+1$ weights. We assume that the temporal correlation varies according to a normalized Gaussian distribution $N(0,m)$ (i.e. the largest value in distribution at mean 0 is 1) around an identified false positive weight. Note that the sampling process is carried out once for an entire execution of RAMP since $2m+1$ weights are updated at each training cycle. Additionally α is constructed by sampling values from the distribution starting from $-m$ to $+m$ with unit step size such that $\alpha[1]$ and $\alpha[2m+1]$ will have the smallest values and $\alpha[m+1]$ will have the highest value of 1.

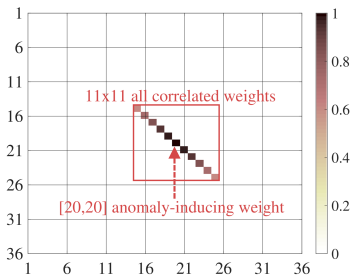


Fig. 4: Illustration of weight selection in Adaptive Training for $d=2, M=40, m=5$ where the sub-sequence with minimum relative distance for each feature is $[20,20]$ and the spread of training bias α.

Example. We use Figure 4 to illustrate weight selection and training bias in the adaptive training module. The total weight space for $d=2, M=40$ and $m=5$ is shown on a 2D lattice. Note that there are $(M-m+1)=36$ sub-sequences for each dimension resulting in a total of $(M-m+1)^d = 36^2 = 1296$ possible keys indexing the weights. Assuming that an anomaly is raised due to the 20^{th} sub-sequence for each dimension, the actual key indexing the anomaly-induced weight is $36^0 \times (20-1) + 36^1 \times (20-1) = 703$, but for illustration purposes, it is shown as $[20,20]$ in Figure 4. In RAMP, each sub-sequence is correlated to m other sub-sequences both in the past and in the future (i.e. $2m+1$ sub-sequences). The $(2m+1)^d = 11 \times 11 = 121$ grid in Figure 4 shows all the weights that are correlated with the weight computed using $[20,20]$. However, updating all these weights incurs an overhead exponentially increasing with dimension d. To address this issue, RAMP picks the most correlated $2m+1$ weights instead of all $(2m+1)^d$ weights. In Figure 4, these weights are shaded at locations $[[15,15],[16,16],...,[25,25]]$, which are centered at the anomaly-inducing weight.

For the training biases, the weights at [16,16] and [25,25] are updated the least due to biases $\alpha[1]$ and $\alpha[11]$, respectively. The anomaly-inducing weight at [20,20] is updated the most due to bias $\alpha[6]$. The variation of the training biases is demonstrated by the color intensity in Figure 4.

C. Human-in-The-Loop Training

RAMP checks whether there are human feedback every M time steps; any human feedback invokes the *Human-in-the-loop training* module, which is shown in Algorithm 3. The human feedback includes the lists of time step indices

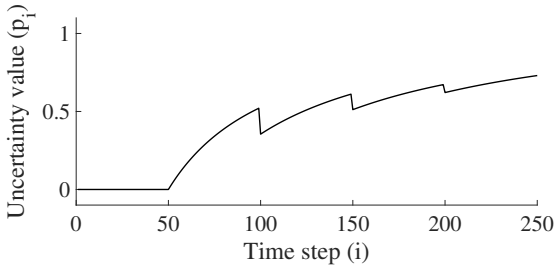


Fig. 5: Change in p_i with i where $M = 50$ and $b = 0.5$

with false positives U_{FP} and true positives U_{TP} among the previous M time steps.

Procedure `HumanInTheLoopTraining`

Data: the time step i, the lists of false positive indices U_{FP} and true positive indices U_{TP} marked from the previous M time steps, the state variable K_{i-M} at step $i - M$, the sub-sequence length m, the window size for periodic user feedback M, the record R and H used for training, the number of dimensions d, and the weights W.

1 $K_i = K_{i-M} + [i - K_{i-M}]\frac{|U_{FP}|}{M}$

2 **for** $i_{TP} \in U_{i,TP}$ **do**

3 $key = 0$

for $j = 1$ *to* d **do**

4 $key \mathrel{+}= (M-m+1)^{j-1} \cdot (R[j, i_{TP}\%M+1]-1)$

end

5 $W[key] = \frac{2W[key]}{H[i_{TP}\%M+1]}$

end

6 **return** $[K_i, W]$

Algorithm 3: *Human-In-The-Loop Training* procedure

Recall that in Eq. (2), state variable K_i is used to calculate the uncertainty function p_i. The human-in-the-loop training procedure first updates K_i according to the formula $K_i = K_{i-M} + [i - K_{i-M}]\frac{|U_{FP}|}{M}$ (Line 1). The intuition here is that, if a large fraction of the previous M time steps has raised false alarms, then we decrease the confidence in the model prediction results by forcing a sudden drop in uncertainty function p_i. Figure 5 illustrates the change in p_i with i when $b = 0.3$ and user feedback is given every $M = 50$ time steps. Here p_i starts with a value of 0 up to $i = 50$ and increases gradually (i.e. $p_{i\to\infty} = 1$). At every 50-th time step we assume that a user marks a list of false positive indices among the past 50 time steps; updating K_i at these time steps causes sudden drops in p_i as shown in Figure 5.

In addition to updating the state variable K_i, the human-in-the-loop training module also updates the weight values that may have been erroneously trained in the past. As RAMP reduces the value of each anomaly-inducing weight to avoid high false alarm rates without human feedback (see Line 9 in Algorithm 2), RAMP rectifies these weights if their corresponding anomalies are verified to be true positives by human users. Given the user-provided set $U_{i,TP}$, which includes the time step indices of true positives, the keys indexing their corresponding anomaly-inducing weights in hash table W are calculated in Lines 3-4; these weights are then updated in a similar manner as in Eq. (1), except that p_i = 0 because we assume the user feedback to be the ground truth (Line 5). Moreover, for an anomaly-inducing weight, its corresponding training bias α is always 1. Thus, the effect of weight updating is that a future use of the new weight on the same unweighted anomaly score as in time step i_{TP} should result in a weighted anomaly score of exactly 2θ.

Because RAMP remembers M past computed results, the total number of time step indices marked by the user is at most M (i.e., $max(|U_{TP}|) \leq M$). The time complexity of Algorithm 3 is $\mathcal{O}(Md)$.

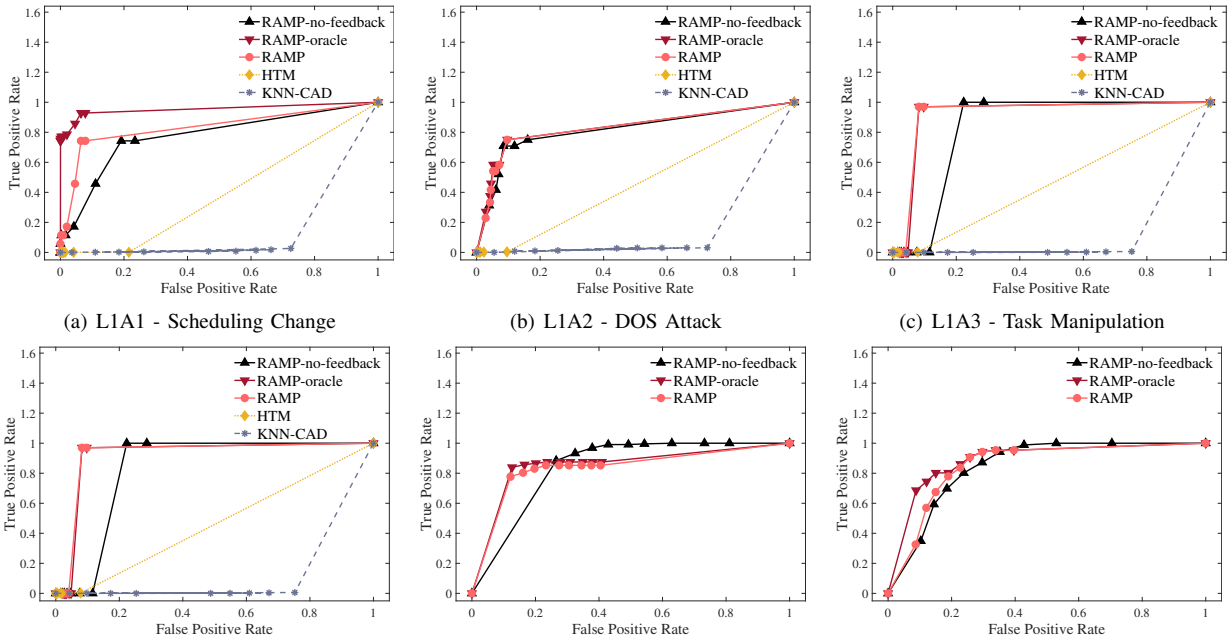
VI. Experimental Results

RAMP was implemented using Python. We have measured the performance of RAMP using the execution logs of single and multiple workflows, collected by the DATAVIEW system running on Amazon EC2 VMs.

In our multiple workflow experiments, we consider the interleaved execution of three workflows – Ligo [7], Wordcount [9] and Diagnosis Recommendation [4]. At any given point in time, only one of the three workflows runs on EC2 VMs. All workflow executions are recorded on a single log instance and no indication is given to anomaly detection models with respect to which log entry corresponds to which workflow. We conducted such experiments to test the robustness of RAMP and its ability to handle noisy data. For each anomaly type, we executed the workflows for six hours and collected the logs, which contain around 5000 data points in the time series. The values for m, M, b, θ are provided by an external user and are used as hyper-parameters in RAMP. The parameters used for performance comparison are $m = 10, M = 200, b = 0.8$ for Level-1 anomalies, and are $m = 5, M = 200, b = 0.8$ for Level-2 adversarial attacks.

A. Performance Comparison: Anomaly Detection

This section presents the performance results of RAMP on detecting Level-1 and Level-2 attacks. We compare three RAMP versions based on the extent of user feedback given, namely *RAMP* (RAMP with both adaptive and human-in-the-loop training), *RAMP-no-feedback* (RAMP with only adaptive training), and *RAMP-oracle* (RAMP with feedback from an all-knowing oracle). *RAMP-no-feedback* is unable to rectify any erroneous training or update its internal state due to the lack of user feedback. *RAMP-oracle* extends Algorithm 3 as follows: for both true positives and false negatives identified by the oracle, *RAMP-oracle* performs weight updating as done in Lines 3-5. In addition, RAMP is compared with two other real-time machine learning models – Hierarchical Temporal Memory (HTM) [3] and KNN-CAD [8] – which are available in the open source Numenta Anomaly Benchmark (NAB) [1]. Among all the machine learning models in NAB, these two models were shown to have good performance in detecting anomalies in scientific workflows [21]. As the anomaly detection models in NAB are designed for univariate time series, we consider one dimensional data input where the dimension most affected by a given anomaly type is fed into all the models for



(a) L1A1 - Scheduling Change (b) L1A2 - DOS Attack (c) L1A3 - Task Manipulation

(d) L1A4 - Workflow Manipulation (e) L2A1 - Workflow Manipulation with Scheduler Change (f) L2A2 - Task Manipulation with DOS Attack

Fig. 6: Receiver Operator Characteristics (ROC) for Interleaved Scientific Workflow Anomalies. Each data point is generated between Threshold (θ) values from $0-1$ with a step size of 0.1 where the highest θ of 1 is at the furthest left.

	Anomaly Type	Anomaly Detection Models				
		RAMP-no-feedback	RAMP-oracle	RAMP	HTM	KNN-CAD
Int.	L1A1 - Scheduler Change	0.7745	0.9538	0.8352	0.4927	0.1498
	L1A2 - DOS Attack	0.8164	0.8315	0.8296	0.4540	0.1516
	L1A3 - Task Manipulation	0.8303	0.9200	0.9236	0.4623	0.1265
	L1A4 - Workflow Manipulation	0.9824	0.9784	0.9645	0.4798	0.1348
	L2A1 - Adversarial Attack	0.8417	0.8528	0.8360	-	-
	L2A2 - Adversarial Attack	0.8474	0.8474	0.8538	-	-
Non-Int.	L1A1 - Scheduler Change	0.8708	0.9172	0.8625	0.4706	0.3492
	L1A2 - DOS attack	0.9620	0.9653	0.9653	0.2160	0.3414
	L1A3 - Task Manipulation	0.9968	0.9968	0.9968	0.3367	0.1695
	L1A4 - Workflow Manipulation	0.9963	0.9989	0.9989	0.4440	0.1674

Table II: Area Under the Curve (AUC) results for Reciever Operator Charachteristics (ROC) for Interleaved (Int.) and Non-Interleaved (Non-Int.) workflows.

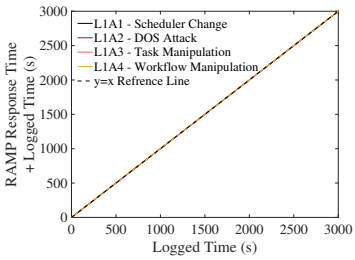


Fig. 7: RAMP Response Time

Level-1 anomaly situations. As Level-2 adversarial anomalies affect multiple dimensions simultaneously, HTM and KNN-CAD models were not used in performance evaluation in these instances.

1) Level-1 Anomalies: Figures 6(a)–6(d) give the Receiver Operator Characteristics (ROC) for the Level-1 anomalies due to Scheduler change, DOS attack, Task Manipulation and Workflow Manipulation, respectively, for interleaved execution of three workflows. In each figure, the three versions of RAMP are compared against HTM and KNN-CAD. We observe from the figures that all the RAMP versions have significantly lower false positives and higher true positives than HTM and KNN-CAD. On average considering the Area Under Curve (AUC) results in Table II for anomalies L1A1-L1A4, RAMP shows an 46.62% and 84.14% increase in AUC compared to HTM and KNN-CAD, respectively. We attribute this increase to the adaptive training module, which adjusts model weights to avoid reporting repetitive anomalies.

Figure 6(a) shows that, at the same false positive rate level, *RAMP-oracle* has higher true positive rates than the other models on L1A1. In addition, *RAMP-no-feedback* has slightly higher false positive rates than the others. We note similar results with respect to other anomaly situations (Figures 6(b)-6(d)). However, when comparing the AUC results of three RAMP versions (Table II), the performance of RAMP is close to that of *RAMP-oracle* and *RAMP-no-feedback* by a difference of $\pm 5\%$.

In single workflow executions, we note similar performance results where all RAMP versions greatly outperforms HTM and KNN-CAD. Due to lack of space, we present only the AUC results for single workflow execution in Table II. As single workflow execution removes the additional noise with regard to anomaly detection, RAMP has increased performance in all anomaly situations (Average Level-1 RAMP AUC increases from 0.8883 to 0.9959).

We observe that the noise caused by interleaved workflow executions has a higher impact on inter-workflow anomalies L1A1 and L1A2 than intra-workflow anomalies L1A3 and L1A4. Inter-workflow anomalies occurring to a given workflow instance can interfere with other interleaving workflows.

RAMP has slightly lower AUC for inter-workflow anomalies, as shown in Figures 6(a) and 6(b), where the maximum true positive rate for RAMP is around 0.8. In contrast, the performance with intra-workflow anomalies is not affected by interleaving workflows because they are local to a single workflow instance.

2) Level-2 Adversarial Anomalies: Figures 6(e) and 6(f) give the performance results of all RAMP versions for Level-2 anomalies. As expected, even though their AUCs decrease slightly, all three RAMP models perform well in adversarial situations (Table II). *RAMP-no-feedback* has its false positive rates increased more significantly than the other two RAMP variants along with a slight increase in its true positive rates. We further note that the performance of RAMP is close to that of *RAMP-oracle*, suggesting that RAMP performs robustly even in adversarial situations.

The above results indicate that RAMP is capable of identifying drastically different anomaly types even without explicit prior definition of possible anomaly situations.

B. Execution Performance: RAMP Response Time

Figure 7 shows the response time of RAMP in Level-1 anomaly scenarios for the first $3,000$ seconds. The x-axis shows the relative time at which DATAVIEW records a log entry; the average rate is 3.4368 seconds per log entry. The y-axis shows the sum of the RAMP response time and the relative logging time by DataView. The figure shows that for all anomaly scenarios the time closely follows the $y = x$ reference line. More specifically, the execution time for RAMP inclusive of adaptive training is approximately 0.0118 seconds on average, which is about 0.0034 times the data stream speed. The results clearly demonstrate the real-time effectiveness of RAMP, where the time taken to detect an anomaly is negligible.

C. Space Complexity: Sparsity of Modified Weights

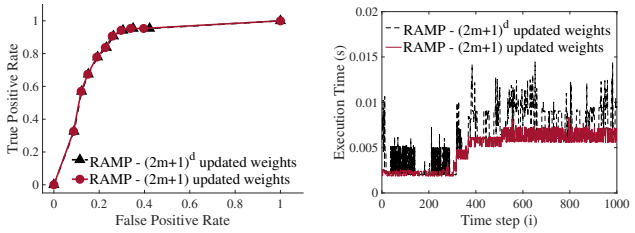
As noted in Section V-A, the total number of possible weights is $(M-m+1)^d$, a value that exponentially increases with d. To address this issue, we propose to store only weights modified by RAMP in a hash table. Our rationale behind this design choice is the sparsity of the number of weights updated with respect to the total possible weights. Table III shows the fraction of the weights modified by RAMP (i.e., $\frac{\#ModifiedWeights}{\#TotalPossibleWeights}$) for each respective workflow in the interleaved scenario for L2A1 where $M = 200, m = 5$ and $d = 3$. The total number of possible sub-sequence combinations (i.e. weights) for each workflow is $(M - m + 1)^d = 196^3 = 7529536$. However the number of updated weights are only 131, 496, and 250 for Diagnosis Recommendation, Ligo, and Wordcount workflow, respectively.

Workflow Name	#Modified Weights	#Total Possible Weights	Fraction
Diagnosis Recommendation	131	7529536	1.7398×10^{-5}
Ligo	496	7529536	6.5874×10^{-5}
Wordcount	250	7529536	3.3203×10^{-5}

TABLE III: Number of Modified Weights

D. Time Complexity: Adaptive Training Heuristics

As described in Section V-B, the adaptive training module uses an optimization heuristic to update only $2m + 1$ weight values at each step instead of $(2m + 1)^d$ weights in complete training. Figures 8(a) and 8(b) show the ROC results and the execution time taken, respectively, for the first 1000 input sub-sequences received for both the complete training and the optimized training cases. Figure 8(a) shows that the two training schemes lead to similar numbers of true positives and false positives. This shows that the optimization heuristic has little effect on the anomaly detection accuracy. Figure 8(b) shows that the optimized training method is about 2.5 times faster than the complete training one. It is noted that there is a sudden increase in execution time after about 300 and 400 time steps for both training methods. This is due to the introduction of the interleaving workflows into DATAVIEW.



(a) ROC (b) Execution time

Fig. 8: Performance of optimization heuristics used in adaptive training (see Section V-B)

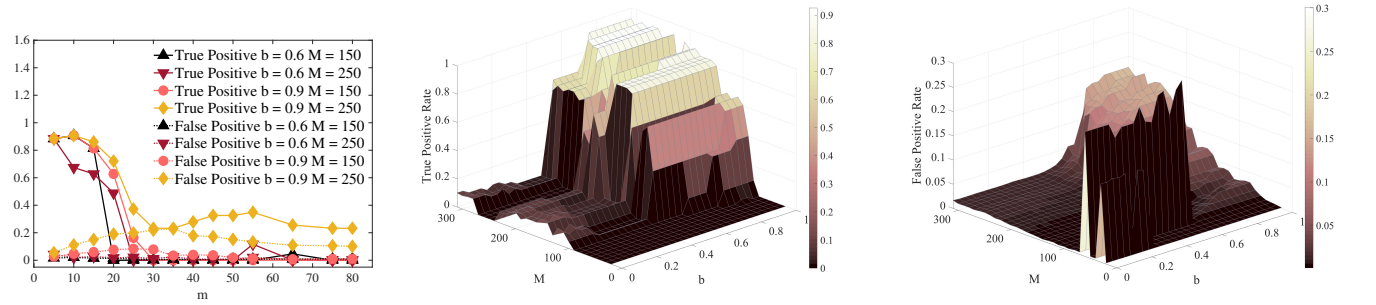
E. Hyper-parameter Tuning

In this section, we aim to provide insight into hyper-parameter tuning in RAMP as well as how each parameter affects the performance of RAMP. Figure 9 shows the impact of parametric variations of m, b, M with a threshold of 0.2 for the L1A4-workflow manipulation anomaly.

We begin by varying the parameter m, which specifies the length of a sub-sequence in Matrix Profile. Figure 9(a) shows the variation of both the true positive rates and the false positive rates under a combination of parameters $b \in \{0.6, 0.9\}$ and $M \in \{150, 250\}$. The figure shows that when $m = 10$, the true positive rate is high. The false positive rate shows limited fluctuations throughout all values of m. The variation of true positive rates and false positive rates is consistent over different combinations of parameters b and M, indicating that parameter m can be tuned independently of the others.

As seen in Eq. (2), the computation of the uncertainty function hinges upon both parameter b and K_i, the latter of which depends on parameter M (Line 1 in Algorithm 3). Figure 9(b) plots the variation of the true positive rate for anomaly type L1A4 (Workflow manipulation) under 900 combinations of b and M. Here we use 30 equal-spaced values of b between 0 and 1, and 30 equal-spaced values for M between 0 and 300. Similarly, Figure 9(c) depicts the false positive rates under the same combinations of parameters b and M. In both figures, the value of m is set to be 10.

Both Figures 9(b) and 9(c) tell us that the anomaly detection performances under different combinations of parameters b



(a) Impact of sub-sequence length (m) on anomaly detection performance

(b) Impact of bias (b) and feedback period (M) on true positive rates

(c) Impact of bias (b) and feedback period (M) on false positive rates

Fig. 9: Impact of parametric change on anomaly detection performance

and M exhibit high-level correlation, suggesting that we can develop a heuristic to identify regions that lead to high true positive rates and low false positive rates. Figure 9(b) shows that regions when $M \leq 50$ tend to have low true positive rates. This is because for small M values, RAMP is unable to create a large enough set for the training base $\mathbf{T}^{'}$. Additionally, RAMP also has low true positive rates when b is smaller than 0.4. For smaller values of b, the uncertainty value converges to 1 more slowly. Hence, when choosing M and b, we should avoid $M \leq 50$ and low $b \leq 0.4$ to improve true positive rates. On the other hand, the false positive rates become high when both M and b are high, or both M and b are small. Combining all these observations, it seems reasonable to choose M and b from the regions of $M \in [150, 300]$ and $b \in [0.4, 0.6]$, or $M \in [50, 150]$ and $b \in [0.5, 1]$.

VII. Related Work

Real-time anomaly detection: Hierarchical Temporal Memory (HTM) is an anomaly detection model designed to replicate the neocortex of mammals and how the neurons learn and predict [3]. While HTM has been used in many real-time anomaly detection applications (e.g [1], [6], [21]), it is limited by its ability to only interpret univariate time series data. Bayesian Online Checkpoint Detection (BOCD) [2] is capable of identifying anomalies in streaming data. However it assumes that the underlying distribution of data is already known. Unlike BOCD, KNN-CAD [8] is non-parametric and probabilistic, where it uses the density and the distance based nearest neighbour algorithm for anomaly detection. Relative Entropy [26] is another lightweight model which uses Turkey and relative entropy statistics. EXPoSE [24] is an anomaly detection model that has the ability to handle multidimenional data. However, as shown in NAB [1] benchmarks, EXPoSE performs poorly when datasets are in moderate size and are univariate.

Anomaly detection in distributed systems: Anomaly detection models have been extensively used in the domain of distributed systems, ranging from preventing DOS attacks [19] to detecting anomalous usage in VMs in the cloud [11]. Various Machine Learning models have been proposed in this application domain, including e.g. the non-parametric clustering model [29], the Support vector Machine model [20], and the use of Baysian Classifiers and tree augmented networks to identify anomalies in distributed systems [25]. Recent work in [12] uses deep learning to detect anomalies in system logs. However, none of the above models are real-time models.

Anomaly detection in scientific workflows: Samark et al. [23] introduce a Naive Bayes model to predict the likelihood of a job success or failure in scientific workflows. [22] proposes an unsupervised model based on K-means clustering that detects hard anomalies (eg. job failure, missing input) and soft anomalies (eg. prolonged execution time) in an online manner. [16] aims to identify time periods where majority of anomalies occur, with the assumption that the workflow failure distribution is a Poisson process. Gaikwad et al. [14] propose a framework to detect performance anomaly w.r.t. the execution time in scientific workflows. Rodriguez et al. [21] conducts a similar study w.r.t performance anomalies in scientific workflows, with the sole intent on investigating the applicability of existing real time machine learning models (eg. HTM, KNN-CAD, Relative Entropy) for anomaly detection. The above work focuses on performance anomalies and none of them considered anomalies resulted from attacks. In addition, none of the above work identify anomalies in multivariate time series, while RAMP identifies anomalies in both univariate and multivariate time series.

VIII. Conclusions

This work is aimed at enhancing existing scientific workflow platforms with new capabilities of misbehavior detection. We develop RAMP, a novel real time anomaly detection model that takes multivariate streaming data input from DATAVIEW logs and produces anomaly alarms in real time. We compare the performance of RAMP with that of two state-of-the-art real-time anomaly detection models. Our results show that RAMP has superior performance while achieving real-time responsiveness for both interleaving and non-interleaving workflows in a variety of anomaly situations.

Acknowledgement: This work is supported in part by the National Science Foundation under grant OAC-1738929.

References

[1] Numenta, https://github.com/numenta/nab, 2019.

[2] R. P. Adams and D. J. MacKay. Bayesian online changepoint detection. *arXiv:0710.3742*, 2007.

[3] S. Ahmad, A. Lavin, S. Purdy, and Z. Agha. Unsupervised real-time anomaly detection for streaming data. *Neurocomputing*, 262:134–147, 2017.

[4] I. Ahmed, S. Lu, C. Bai, and F. A. Bhuyan. Diagnosis recommendation using machine learning scientific workflows. In *IEEE Congress on Big Data*, pages 82–90, 2018.

[5] S. D. Anton, L. Ahrens, D. Fraunholz, and H. D. Schotten. Time is of the essence: Machine learning-based intrusion detection in industrial time series data. In *ICDM Workshops*, pages 1–6, 2018.

[6] V. Berger. Anomaly detection in user behavior of websites using hierarchical temporal memories: Using machine learning to detect unusual behavior from users of a web service to quickly detect possible security hazards., 2017.

[7] D. A. Brown, P. R. Brady, A. Dietz, J. Cao, B. Johnson, and J. McNabb. A case study on the use of workflow technologies for scientific analysis: Gravitational wave data analysis. In *Workflows for e-Science*. 2007.

[8] E. Burnaev and V. Ishimtsev. Conformalized density-and distance-based anomaly detection in time-series data. *arXiv:1608.04585*, 2016.

[9] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[10] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good. The cost of doing science on the cloud: the montage example. In *ACM/IEEE conference on Supercomputing*, page 50, 2008.

[11] F. Doelitzscher, M. Knahl, C. Reich, and N. Clarke. Anomaly detection in iaas clouds. In *International Conference on Cloud Computing Technology and Science*, volume 1, pages 387–394, 2013.

[12] M. Du, F. Li, G. Zheng, and V. Srikumar. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1285–1298. ACM, 2017.

[13] D. Fanelli. How many scientists fabricate and falsify research? a systematic review and meta-analysis of survey data. *PloS one*, 4(5), 2009.

[14] P. Gaikwad, A. Mandal, P. Ruth, G. Juve, D. Król, and E. Deelman. Anomaly detection for scientific workflow applications on networked clouds. In *International Conference on High Performance Computing & Simulation*, pages 645–652, 2016.

[15] C. A. Goble, J. Bhagat, S. Aleksejevs, D. Cruickshank, D. Michaelides, D. Newman, M. Borkum, S. Bechhofer, M. Roos, P. Li, et al. myexperiment: a repository and social network for the sharing of bioinformatics workflows. *Nucleic acids research*, 38(suppl 2):677–682, 2010.

[16] D. Gunter, E. Deelman, T. Samak, C. H. Brooks, M. Goode, G. Juve, G. Mehta, P. Moraes, F. Silva, M. Swany, et al. Online workflow management and performance analysis with stampede. In *International Conference on Network and Service Management*, pages 1–10, 2011.

[17] A. Kashlev and S. Lu. A system architecture for running big data workflows in the cloud. In *2014 IEEE International Conference on Services Computing*, pages 51–58. IEEE, 2014.

[18] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski. Cost-and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds in: Proceedings of the international conference on high performance computing, networking, storage and analysis, 22, 2012.

[19] A. Navaz, V. Sangeetha, and C. Prabhadevi. Entropy based anomaly detection system to prevent ddos attacks in cloud. *arXiv:1308.6745*, 2013.

[20] H. S. Pannu, J. Liu, and S. Fu. A self-evolving anomaly detection framework for developing highly dependable utility clouds. In *IEEE GLOBECOM*, pages 1605–1610, 2012.

[21] M. A. Rodriguez, R. Kotagiri, and R. Buyya. Detecting performance anomalies in scientific workflows using hierarchical temporal memory. *Future Generation Computer Systems*, 88:624–635, 2018.

[22] T. Samak, D. Gunter, M. Goode, E. Deelman, G. Juve, G. Mehta, F. Silva, and K. Vahi. Online fault and anomaly detection for large-scale scientific workflows. In *International Conference on High Performance Computing and Communications*, pages 373–381, 2011.

[23] T. Samak, D. Gunter, M. Goode, E. Deelman, G. Juve, F. Silva, and K. Vahi. Failure analysis of distributed scientific workflows executing in the cloud. In *8th international conference on network and service management*, pages 46–54, 2012.

[24] M. Schneider, W. Ertel, and F. Ramos. Expected similarity estimation for large-scale batch and streaming anomaly detection. *Machine Learning*, 105(3):305–333, 2016.

[25] Y. Tan et al. Online performance anomaly prediction and prevention for complex distributed systems. 2012.

[26] C. Wang, K. Viswanathan, L. Choudur, V. Talwar, W. Satterfield, and K. Schwan. Statistical techniques for online anomaly detection in data centers. In *IFIP/IEEE International Symposium on Integrated Network Management*, pages 385–392, 2011.

[27] C.-C. M. Yeh, H. Van Herle, and E. Keogh. Matrix profile iii: the matrix profile allows visualization of salient subsequences in massive time series. In *ICDM*, pages 579–588, 2016.

[28] C.-C. M. Yeh, Y. Zhu, L. Ulanova, N. Begum, Y. Ding, H. A. Dau, D. F. Silva, A. Mueen, and E. Keogh. Matrix profile i: all pairs similarity joins for time series: a unifying view that includes motifs, discords and shapelets. In *ICDM*, pages 1317–1322, 2016.

[29] L. Yu and Z. Lan. A scalable, non-parametric anomaly detection framework for hadoop. In *ACM Cloud and Autonomic Computing Conference*, page 22, 2013.

[30] Y. Zhu, Z. Zimmerman, N. S. Senobari, C.-C. M. Yeh, G. Funning, A. Mueen, P. Brisk, and E. Keogh. Matrix profile ii: Exploiting a novel algorithm and gpus to break the one hundred million barrier for time series motifs and joins. In *ICDM*, pages 739–748, 2016.