2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)
October 11-14, 2020. Toronto, Canada

MA2DF: A Multi-Agent Anomaly Detection Framework

Yohen Thounaojam
Department of Computer Science
The University of British Columbia
BC, Canada
yohenthounaojam@gmail.com

Wiliam Setiawan
Department of Computer Science
The University of British Columbia
BC, Canada
wiliamd.great@gmail.com

Apurva Narayan
Department of Computer Science
The University of British Columbia
BC, Canada
apurva.narayan@ubc.ca

***Abstract*—Time-sensitive safety-critical systems store traces as a collection of time-stamped messages that are generated while a system is operating. Analysis of these traces becomes a key task as it allows one to find faults or errors within a system that is otherwise difficult to discern, especially in complex systems. Furthermore, finding any form of anomalous behaviour becomes critical in time-sensitive and safety-critical systems where a late detection will often lead to dire consequences. Most available approaches are generally used in networking or business process analysis. We focus on creating a lightweight and explainable approach for time-sensitive safety-critical systems.**

By using a set of system traces under both normal and anomalous conditions, our approach attempts to classify whether or not a trace is anomalous. In this work, we introduce MA2DF, Multi-Agent Anomaly Detection Framework, a novel multi-agent based graph design approach for online and offline anomaly detection in system traces. Our approach takes advantage of the timing information between a sequence of events and also the event sequences to learn and discern between normal and anomalous traces. We present two approaches, an offline approach to discern anomalous behaviour by utilizing the event occurrence workflow graph. The second approach is an online streaming algorithm that monitors the sequence of events as they arrive in real-time. This can be used to detect anomalies, find the cause, and improve system resilience. We show how our approach, MA2DF, is superior to other state-of-the-art models. The paper will explore the technical feasibility and viability of MA2DF by utilizing industry strength case study using traces from a field-tested hexacopter.

***Index Terms*—anomaly detection, graph, multi-agent systems, cyber-physical systems, workflow graphs**

I. Introduction

Numerous real-world systems are equipped with the ability to generate system trace/log files that allow users to analyze the system's behaviour. This could provide key insight into the system performance and allow the user to tell whether or not the system is behaving normally. This could be shown in numerous ways in the trace file, for example whether or not a set of events are running within certain time constraints. Analyzing these files becomes a key part of managing an operating system and it is invaluable to improve upon the techniques to do so. Given that these system traces can get exponentially larger as the system size grows, it becomes infeasible to manually analyze these graphs. Thus it is an absolute necessity to create an automated technique to analyze these system traces.

To specify things further, in the context of embedded systems, multiple subcomponents integrate and work together to ensure the system is operating normally. There are standards and requirements for these systems such as the ISO-26262 in automotive systems or the DO178C for airborne systems. Through analyzing system traces, one can provide assurance and conformance with such standards. Furthermore, analyzing system traces allows one to improve the credibility of post-mortem analysis in embedded systems.

The study of anomaly detection has been an ongoing endeavor since the early 19^{th} century [1] in the statistics community. Recently, the study has created numerous anomaly detection techniques across different domains and research committees. These techniques primarily involve finding patterns in data to discern whether or not a system is behaving normally or abnormally. These patterns also allow us to identify key points where the system begins its anomalous behaviour. These techniques are invaluable in the real world and have an extensive application, be it in intrusion detection for cyber-security, fraud detection for credit cards, insurance or health care, intrusion detection for cyber-security, military surveillance for enemy activities or fault detection in safety-critical systems.

Workflow graph based anomaly detection is a subsidiary technique in anomaly detection. This method focuses on detecting attacks as a divergence from the traditional system behaviour. One key difference in graph based approaches compared to machine-learning based approaches is that it requires a hand-crafted or semi-automated approach to system behavioral modelling that captures legitimate system behaviour in some formal way. In order to fulfill this, many approaches choose to use a workflow graph as it minimizes the number of false alarms caused by normal yet previously unseen behaviour. The downside of these kinds of approaches is that developing such graphs could be tedious and a time-consuming endeavor. The method we propose aims to minimize the complexity to

978-1-7281-8526-2/20/$31.00 ©2020 IEEE

generate such graphs whilst also minimizing the rate of false alarms in the system.

In this work, we present a novel multi agent greedy approach to workflow graph construction from just the event and timestamp information retrieved from system traces. This approach can then be used for offline and online anomaly detection for safety-critical real-time systems. The key contributions of this paper include:

- An evolved multi agent greedy approach to workflow graph construction from system traces using the sequence of events and the inter-arrival time between these events.
- An offline anomaly detection technique which compares the workflow graphs generated from clean and anomalous systems traces.
- An online monitoring approach using real-time system traces for online anomaly detection.

The following sections of the paper is organized in the following manner: Section II will discuss related work in context to anomaly detection from system traces, Section III will explain our approach for workflow graph construction, offline anomaly detection, and online anomaly detection. Section IV explains the experimental setup, the data set used from QNX operating system, and Section V will provide results, conclusions and direction for future work.

II. Related Work

Anomaly detection is an active area of research that finds application in a diverse range of applications from safety-critical systems to business process analysis. Recently, an encyclopedic review has been published on anomaly detection that covers a diverse range of applications [2].

Anomaly detection approaches have emerged from a variety of fundamental fields specifically in the context of event driven systems. For instance, in [3], authors use an information theoretic approach and extrapolate measures such as entropy, conditional entropy, information gain etc. for anomaly detection. In [4] they employ a graph based anomaly detection approach. In [4] they perform the task of anomaly detection by calculating the regularity of the graph, which is used to identify aberrations from normal behavior. One major limitation of such an approach is that the availability of data in graph structure is not easily available. We, therefore, address this issue of developing the workflow or system processes in the form of a graph that can eventually be used for anomaly detection.

Detecting anomalies using log files is a crucial and extremely dynamic field of study for networks. An exhaustive review of a diverse set of approaches for network anomaly detection and/or network intrusion detection techniques has been given in [5].

Anomaly detection in hardware resources such as hard drives is critical to ensuring that data is safe and useful. A novel framework that computes meaningful storage performance metrics from low-level trace events generated by LTTng was introduced in [6]. Their approach is focused on the system state to infer the sub-system's performance. An intuitive visualization schema is provided that grants a myriad of graphical views that represent the collected information in a convenient way. Console log analysis is a key ingredient for analysis of Linux based systems where [7] develop a reachability graph to analyze the reachable relations of log files. Information retrieval techniques play a key role in sequential data analysis. Additionally, a probability suffix tree approach helps extricate significant statistical properties of the sequences.

Root cause analysis is a key differentiating factor between anomaly detection techniques. One such work is [8] where they use localized anomalously invoked methods and their physical locations by leveraging request trace logs. They have a two-step process of clustering and dimensionality reduction followed by similarity analysis using Jensen-Shannon divergence. As highlighted earlier, business process log files are similar in nature to the system traces and detecting anomalies in them is a very similar task. In [9] a new model using the dynamic bayesian networks with numerical attributes and functional dependencies is introduced to model the system log's behavior. An interesting feature of this approach is the decomposition score that indicates the root cause of the anomalies.

Anomaly detection through graphs is not a new endeavor by any means. This method has been explored several times before, for example in [10], authors propose an approach for mining causality and diagnosing root cause analysis that uses knowledge graph technology and a causal search algorithm. They tested their approach by using a native cloud application. In another case, work was done on subjects closely related to runtime anomaly detection for embedded systems. This work [11] used on-chip hardware to non-intrusively monitor system execution through trace port of the processor and detect malicious activities at runtime. They also measure the timing information for the flow of events of the system. There has also been work done where they use dependency graphs as solution techniques [12]. Finally, a survey on graph based anomaly detection techniques has been done and shown in [13].

Pattern mining is a well researched area and is applicable in a wide range of domains for data mining from system traces [14], [15], system behaviour identification [16], automated reasoning or business process log. These frequent pattern mining approaches help in interpreting system behavior and the development of confident anomaly detection frameworks. An overview of the current work in frequent pattern mining is provided in [17]. Authors in [17] also discuss promising research directions that can use these approaches. Correspondingly, temporal sequence mining approaches have been explored for application in anomaly detection such as [18].

Our work focuses on workflow based graph generation and utilize them for anomaly detection and runtime monitoring of a wide variety of systems. The next section will provide further

details of our proposed framework MA2DF.

III. MA2DF: A Multi-Agent Anomaly Detection Framework

We formalize the problem of workflow graph inference in terms of the inference of a weighted graph which captures the sequential relationship between events in the system trace and the inter-arrival time between the events.

We formulate the problem of learning the workflow graph topology as the inference of a graph $G = (V, E)$, where the vertices $V = v_i$ represent the unique events in the system trace, and the edges $E = e_{i,j}$ represent the connectivity between them; an edge $e_{i,j}$ denotes a path from the event v_i to an event v_j. The sources of event occurrences are modelled as some number N of agents moving asynchronously through the graph. Each agent generates an observation every time it visits a vertex. The input to the problem is an ordered list of observations $O = o_t$, each of which is identifiably generated by one of the events; i.e. each $o_t \in [1, V]$. The goal is to find the correct underlying graph G explaining this observational sequence.

A. Multi-Agent Graph Construction

The key idea behind our approach is to find the smallest graph that successfully explains the observed data. Leaving aside for the moment the actual implementation details, let us consider this idea in more depth by proposing the existence of an algorithm A that takes as an input the assumed number of agents N in the environment and the observational sequence and returns as output the smallest graph consistent with the observations.

Our algorithm A considers each of the possible trajectories that could be taken by these N agents given the observational sequence and then selects the trajectory set that requires the smallest number of inter-vertex traversals. The algorithm then returns the graph populated only with edges that correspond to the inter-vertex traversals required by this chosen trajectory set.

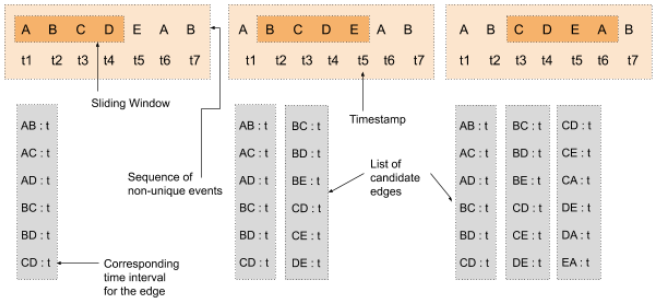

Fig. 1. Visualization of the sliding window approach

Considering a certain number of agents, N, traversing the events of a system trace, varying graphs can be formed. The graph varies with the number of agents as we employ a sliding window approach [19]. The example in Figure 1 uses the sliding window approach over the series of events with an assumed number of agents as $N = 4$. This way, we form $e - (N - 1)$ windows, where e is the number of non-unique events. Adopting the same approach, we will form the windows for the timestamp.

Hence we end up with candidate lists $L^1, L^2, ...L^K$ and corresponding timestamps $T^1, T^2, ...T^k$. Then we follow the greedy algorithm:

Algorithm 1: Greedy Algorithm Graph Construction.

```
Result: Weighted Graph
mark all candidate lists as unexplained;
finalGraph; allEdges = [{edge : timeData}];
for list in candidate lists do
    for edge in list do
        timeData = [count, μ, min, max, σ]
        add {edge : timeData} to allEdges;
    end
end
countExplained = 0;
while countExplained != len(windows) do
    edge = edge with highest frequency;
    mark all candidate lists with edge as explained;
    add edge to finalGraph;
end
return finalGraph;
```

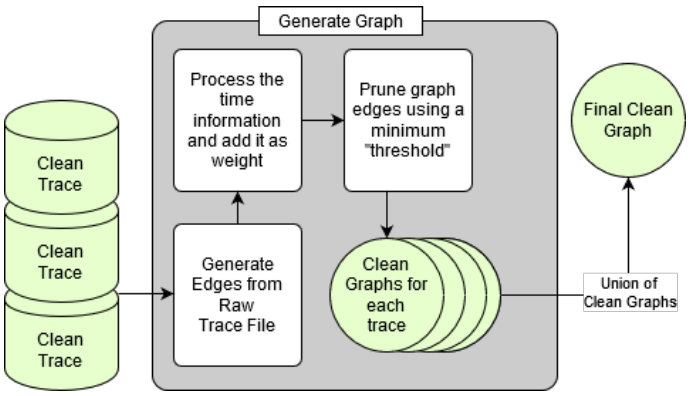

Fig. 2. Workflow Graph Generation

The concept that the simplest solution explaining the data is probably the correct solution has been used successfully in a different version of the topology inference problem [19]. The principle, known as Occam's razor, states, "if presented with a choice between indifferent alternatives, then one ought to select the simplest one." The concept is a common theme

in computer science and underlies a number of approaches in AI; e.g. hypothesis selection in decision trees and Bayesian classifiers.

B. Offline Anomaly Detection

First, a graph of the unknown trace is generated using the methods in Subsection A. To detect anomalies in an already recorded system trace, the following two methods can be employed to check for similarity between the two graphs. The first approach is simple checking if the graphs are Isomorphic to the graph generated using the clean trace. If the two graphs, one clean and the other unknown are isomorphic, then we can simply conclude that they are the same; and hence, the unknown graph can be categorized as not anomalous. If this isn't the case, we measure the Jaccard similarity index between these two graphs. Analyzed on a scale of 0.0 to 1.0, results of 0.0 are the least similar graphs and 1.0 being the most similar. The Jaccard coefficient measures similarity between finite sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets.

Diving deeper into using Jaccard's similarity index, the index value can be set as a threshold. Depending on the system and the insight we get from learning the clean traces, we can set a threshold to classify all traces whose workflow graphs result in a Jaccard's similarity index less than the threshold when compared to the clean traces' workflow graph.

The algorithm considers two inputs, a clean system trace and an uncategorized trace (which will be flagged as anomalous later). Using the approach in Section A, we form workflow graphs of both traces. Hence, we now have a workflow graph that is not anomalous in any way and another workflow graph we are not aware of; both from the same system. To determine whether or not the latter is anomalous, we can compare it using the two criteria; isomorphism and Jaccard similarity index. This is visualised in Figure 3.

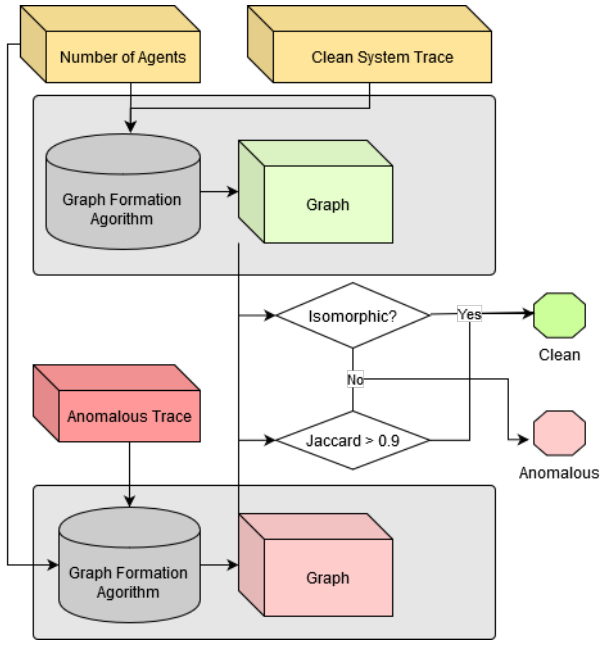

Fig. 3. Offline Anomaly Detection Workflow

C. Online Anomaly Detection

Although detecting anomalous behaviours through an offline approach is insightful, it is usually only utilized for fault diagnosis and root cause analysis. In real-world situations, it would be even more insightful to monitor systems continuously and detect anomalous behaviours as they arise in the system runtime. Thus we propose an online anomaly detection framework.

We developed an online anomaly detection or monitoring approach as shown in Figure 4. In this framework, we traverse through the composed workflow graph from the set of clean traces. As events arrive, we monitor and evaluate the system behaviour taking into close consideration two key aspects. The first is whether or not a path exists from the current node to the following event node. The second aspect is whether or not any event transition comes without breaking the inter-arrival time constraint. In this version, anomaly detection will be done through checking whether or not subsequent events exist as an edge in the clean graph. The threshold we use to determine whether or not a system is anomalous is obtained during the training process. We use two resilience parameters, γ_e and γ_t. γ_e is the threshold for identifying a trace as anomalous due to an incorrect sequence of events. γ_t on the other hand is the threshold for anomalous behaviour when the system fails to meet the inter-arrival time bounds obtained earlier though the training process.

For our purposes, we prioritize γ_e over γ_t as failing to meet a certain sequence of events could lead to catastrophic outcomes leading to system failure. On the other hand, failing to meet time constraints may simply be an indicator of precursor to system failure. Thus this implies that by using both of these parameters, we can establish system resiliency. Through constant monitoring of both these parameters on a system, one can raise a flag on a system if they see any of these parameters being violated frequently. For establishing resilient systems in the case study, a comprehensive analysis is presented.

Below we show a pseudo-code for our online anomaly detection framework. For every time the two resilience parameters are not met, the event info is added to an Anomaly Stack of preferred size. Depending on the system and situation in hand, the parameter must be changed. Simulating a real-time system, the algorithm below iterates over a recorded system trace, treating each event as a new event logged in real-time.

In the pseudo-code below, cleanGraph is the graph obtained from clean non-anomalous traces, finalGraph is formed by

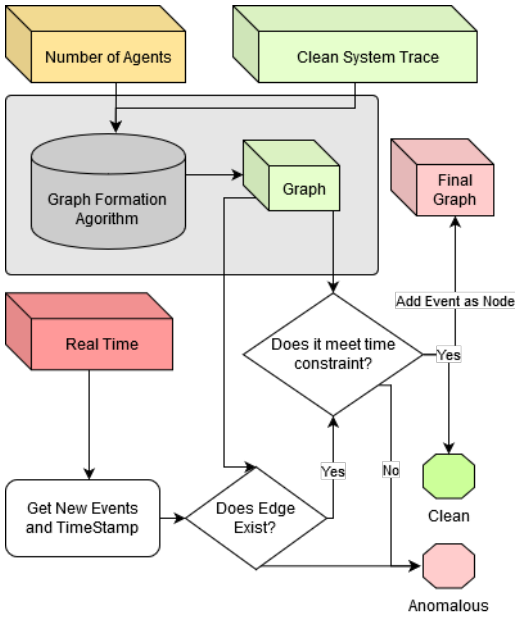

Fig. 4. Online Anomaly Detection Workflow

the unknown trace, unknownTrace. An anomalyStack is used to keep count of the times an edge violated the resilience parameters.

As seen in Fig. 4, the algorithm takes two inputs, a clean system trace and a real-time log of the system. This is accompanied by the number of agents as a tuning parameter. As we get new events with timestamps, we check if the edge exists in the graph generated by the clean trace. Then, we check if the time constraint is met. If the edge does not exist or the time constraint is not met, then an anomaly flag is raised.

Algorithm 2: Online Anomaly Detection.

```
Result:
cleanGraph; finalGraph;
unknownTrace[];
anomalyStack; γ1; γ2;
for event in unknownTrace do
    if γ1 meets threshold or γ2 meets threshold or
     anomalyStack is full then
        Anomaly detected;
    else
        if first event and exists in cleanGraph then
            add event to cleanGraph;
            continue;
        else
        end
        if event exists in cleanGraph edge = edge of
        current and previous events. if edge in
        cleanGraph then
            if time withing min and max then
                add egde to finalGraph;
            else
                γ1 + +;
                add edge to anomalyStack;
            end
        else
            γ2 + +;
            add edge to anomalyStack;
        end
        add edge to anomalyStack;
    end
end
return finalGraph;
```

IV. Experimental Evaluation

In this section, we employ graphs generated from event traces of real-time systems obtained from two industrial case studies to show the usefulness of the framework in Section III.

UAV Case Study

For our case study, we use an unmanned aerial vehicle's (UAV) kernel event traces running a real-time operating system QNX Neutrino 6.4.

Developed at the University of Waterloo, the UAV was flown to carry out real payload-drop and mapping missions in the provinces of Ontario and Nova Scotia, Canada. This required the acquiring of a Special Flight Operating Certificate (SFOC). A sample of the trace used in the experiments is presented in Figure 5.

Developed at the University of Waterloo, the UAV acquired a Special Flight Operating Certificate (SFOC) and flew real payload-drop and mapping missions in Nova Scotia and Ontario, Canada. The trace snippet in Figure 5 shows the sample trace used in our experiments.

The snippet is generated by using the the *tracelogger* and *traceprinter* utilities from the QNX Neutrino real-time operating system. An event type possesses unique values obtained by a combination of the process name `pname` as well as the process class *pclass*. The dataset being used in the experiments has 15 traces obtained from a UAV where each of these traces has a stream of roughly 1 million events. There are four scenarios where anomalies may be detected.

- One scenario runs task that implements a while loop to deplete CPU time.
- Two scenarios implement a job executed every few seconds where the task is scheduled using two different scheduling algorithms (e.g., FIFO and sporadic scheduling).

```
259018352791,13,INT_ENTR::0x00000044
259018354208,14,INT_HANDLER_ENTR::0x00000044
259018357541,15,INT_HANDLER_EXIT::0x00000044
259018368666,18,COMMSND_PULSE_EXE
259018370333,18,COMMSND_PULSE_EXE
259018371583,18,COMMSND_PULSE_EXE
...
259567335041,22,COMMREC_MESSAGE
259567336541,23,KER_EXITMSG_RECEIVEV/14
259567355458,13,INT_ENTR::0x00000044
259567366416,14,INT_HANDLER_ENTR::0x00000044
259567381750,15,INT_HANDLER_EXIT::0x00000044
259567384916,16,INT_EXIT::0x00000044
259567403625,25,COMMREPLY_MESSAGE
```

Fig. 5. Trace snippet from QNX *tracelogger*

- The last scenario corresponds to a normal execution behavior while not complying to the traces generated during training.

Using Algorithm 1, the individual graphs for all the clean traces are generated. With the individual graphs, a union of all the graphs of clean traces is performed to obtain a final single graph. This is followed by two separate experiments for online and offline anomaly detection as shown below:

Offline Anomaly Detection Experiments

To detect anomalies in recorded system traces, we adopt a simple idea of comparing the test trace with a known clean trace's workflow graph. To do so, we generated workflow graphs using Algorithm 1 for each clean trace. Then, a union of all the clean workflow graphs is formed while preserving the time data for each edge. For any edge that occurred more than once, the maximum time interval was stored. Now, to be able to compare this trace with the test trace, a workflow graph of the test trace was generated using Algorithm 1. Once we obtained the two workflow graphs (clean and test), we evaluated the similarity between them.

As outlined in Section III-B, we first check whether the two graphs are isomorphic. If not isomorphic, we check for the Jaccard's similarity index. The experiment consisted of four individual anomalous traces. For evaluating our graph construction approach we decided to vary the number of agents, N to analyze its influence on the similarity with both, anomalous and clean traces. However, independent of the number of agents, the Jaccard's similarity index retrieved was the same for each respective anomalous trace. Four anomalous traces were evaluated and they resulted in a Jaccard's similarity index of 0.55, 0.557, 0.497 and 0.55; irrespective of the number of agents, N. This indicates a hundred percent accuracy in classifying the trace as anomalous. An interesting result was how the number of agents, N, did not affect the result. Generally, this would be an unexpected result; however, it can be concluded that the test trace for this particular system is only sensitive to consecutive events and this consequence was data/case specific. Hence, considering a higher N can be disregarded in this test case.

To check whether the individual clean traces were classified as not anomalous when compared to the clean workflow graph, we performed the same experiment for the clean traces. The technique successfully classified the traces as clean with Jaccard's similarity indices of 0.943, 0.933, 1.0 and 0.946. Our results are promising since, in our industry strength case study, we classified anomalous traces with 100% accuracy. Moreover, the workflow graph generated in the process also enables one to perform a deep fault diagnosis and root cause examination.

Model	Accuracy	Explainability
Markov Chains	99%	Partial
Feedforward Neural Networks	97.3%	No
Bayesian Network	94%	Partial
LSTM	96.5%	No
Random Forests	100%	Yes
MA2DF	100%	Yes

TABLE I
COMPARISON WITH STATE-OF-THE-ART-TECHNIQUES

In Table I the comparison between MA2DF and other state-of-the-art techniques on the same dataset is presented. As evident in the table, the approach we used in MA2DF performs either at-par or surpasses the the state of the art methods. However, MA2DF has other advantages such as it being straightforward, light-weight, and providing explainability or root cause examination which is lacking in most complex models.

Streaming Anomaly Detection Experiments

When dealing with online streaming anomaly detection, the examination of a system's behaviour must be performed in real-time. Safety-critical real-time systems may indicate anomalous behaviors through several behaviors. The first one is an abnormal sequence of events and the second one is failing to meet the time constraints of each event. In order to encompass these two properties, we propose two resiliency parameters γ_e and γ_t that allow us to monitor the behavior in real-time. We obtain bounds for both γ_e and γ_t through our training phase and through the clean traces. γ_e denotes the maximum number of times an event fails to meet the corresponding pairwise event co-occurrence from the workflow graph previously generated. γ_t denotes the maximum number of times the pairwise event co-occurrence sequentially was correct however it fails to meet the time bounds for said pairwise event. We keep a count of both, the maximum sequential count of the events that are expected to occur together but did not and similarly for the inter-arrival time bounds for clean traces. In the case where a trace either goes beyond γ_e and γ_t, a flag is raised to indicate that the system is showing anomalous behaviours. It is observed that in the case of

clean traces, the average value of the two resilience parameters is $\gamma_e = 7$ and $\gamma_t = 1$ respectively. While on anomalous datasets it shows values of $\gamma_e = 202.5$ and $\gamma_t = 1.75$ for each parameters respectively. As shown in the results above, there is a large margin between clean and anomalous trace results, and it is thus a trivial case to determine whether a trace is anomalous or not.

V. Conclusions and Future Work

We demonstrate an approach to generate workflow graphs from system traces using a multi-agent framework. Our work is applicable in the domain of safety-critical systems. Our unified workflow for graph generation and anomaly detection with the ability for explainability provides a unique amalgamation of approaches. The framework can operate in an online and offline mode as per the need. The experiments certify that our work performs well on industry strength use case of a real-time safety-critical system. We also present a comparison of our approach with various well known techniques in Table I, showing how MA2DF outperforms them. We intend to extend this work for incorporating and extracting formal system properties in the form of timed regular expressions or quantitative regular expression for the purpose of complex system modelling and analysis.

Acknowledgment

The authors would like to thank Prof. Sebastian Fischmeister, Electrical and Computer Engineering, the University of Waterloo for providing datasets for use in our work.

References

[1] F. Y. Edgeworth, "Xxii. on a new method of reducing observations relating to several quantities," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 25, no. 154, pp. 184–191, 1888.

[2] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys*, vol. 41, no. 3, 2009. [Online]. Available: https://doi.org/10.1145/1541880.1541882

[3] Wenke Lee and Dong Xiang, "Information-theoretic measures for anomaly detection," in *Proceedings 2001 IEEE Symposium on Security and Privacy. S P 2001*, 2001, pp. 130–143.

[4] C. C. Noble and D. J. Cook, "Graph-based anomaly detection," in *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: Association for Computing Machinery, 2003, p. 631–636. [Online]. Available: https://doi.org/10.1145/956750.956831

[8] H. Mi, H. Wang, Y. Zhou, M. R. Lyu, and H. Cai, "Localizing root causes of performance anomalies in cloud computing systems by analyzing request trace logs," *Science China Information Sciences*, vol. 55, no. 12, pp. 2757–2773, Dec 2012.

[5] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, "Network anomaly detection: Methods, systems and tools," *IEEE Communications Surveys Tutorials*, vol. 16, no. 1, pp. 303–336, 2014.

[6] H. Daoud and M. R. Dagenais, "Recovering disk storage metrics from low-level trace events," *Software: Practice and Experience*, vol. 48, no. 5, pp. 1019–1041, 2018. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2566

[7] L. Bao, Q. Li, P. Lu, J. Lu, T. Ruan, and K. Zhang, "Execution anomaly detection in large-scale systems through console log analysis," *Journal of Systems and Software*, vol. 143, pp. 172 – 186, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121218301031

[9] S. Pauwels and T. Calders, "Detecting anomalies in hybrid business process logs," *SIGAPP Appl. Comput. Rev.*, vol. 19, no. 2, p. 18–30, Aug. 2019. [Online]. Available: https://doi-org.ezproxy.library.ubc.ca/10.1145/3357385.3357387

[10] J. Qiu, Q. Du, K. Yin, S.-L. Zhang, and C. Qian, "A causality mining and knowledge graph based method of root cause diagnosis for performance anomaly in cloud applications," p. 2166, 2020.

[11] S. Lu and R. Lysecky, "Time and sequence integrated runtime anomaly detection for embedded systems," *ACM Trans. Embed. Comput. Syst.*, vol. 17, no. 2, Dec. 2017. [Online]. Available: https://doi-org.ezproxy.library.ubc.ca/10.1145/3122785

[12] T. B. Callo Arias, P. van der Spek, and P. Avgeriou, "A practice-driven systematic review of dependency analysis solutions," *Empirical Software Engineering*, vol. 16, no. 5, pp. 544–586, Oct 2011. [Online]. Available: https://doi.org/10.1007/s10664-011-9158-8

[13] L. Akoglu, H. Tong, and D. Koutra, "Graph based anomaly detection and description: a survey," *Data Mining and Knowledge Discovery*, vol. 29, no. 3, pp. 626–688, 05 2015, copyright - The Author(s) 2015; Last updated - 2015-04-15. [Online]. Available: http://ezproxy.library.ubc.ca/login?url=https://search-proquest-com.ezproxy.library.ubc.ca/docview/1671986357?accountid=14656

[14] A. Narayan, N. Benann, and S. Fischmeister, "Mining specifications using nested words," in *2017 6th International Workshop on Software Mining (SoftwareMining)*, 2017, pp. 9–16.

[15] A. Narayan, G. Cutulenco, Y. Joshi, and S. Fischmeister, "Mining timed regular specifications from system traces," *ACM Trans. Embed. Comput. Syst.*, vol. 17, no. 2, Jan. 2018. [Online]. Available: https://doi.org/10.1145/3147660

[16] A. Narayan and S. Fischmeister, "Mining time for timed regular specifications," in *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, 2019, pp. 63–69.

[17] J. Han, H. Cheng, X. Dong, and X. Yan, "Frequent pattern mining: current status and future directions," *Data Mining and Knowledge Discovery*, vol. 15, no. 1, pp. 55–86, 08 2007, copyright - Springer Science+Business Media, LLC 2007; Last updated - 2014-08-30.

[18] T. Lane and C. E. Brodley, "Temporal sequence learning and data reduction for anomaly detection," *ACM Trans. Inf. Syst. Secur.*, vol. 2, no. 3, p. 295–331, Aug. 1999. [Online]. Available: https://doi-org.ezproxy.library.ubc.ca/10.1145/322510.322526

[19] D. Marinakis and G. Dudek, "Topological mapping through distributed, passive sensors." in *IJCAI*, 2007, pp. 2147–2152.