A Method for Temporal Event Correlation

Robert Harper
Moogsoft Ltd
River Reach, 31-35 High Street
Kingston-Upon-Thames, UK
rob@moogsoft.com

Philip Tee
Moogsoft Inc
1265 Battery St
San Francisco, CA 94111
phil@moogsoft.com

***Abstract*—In recent years, the use of data-driven algorithms has gained significant traction as a method of localizing faults in commercial networks through the analysis of network events.**

While the number of different failure scenarios in any IT infrastructure is potentially unbounded, the fundamental mechanisms of fault propagation and related failure modes are much smaller. A significant component of many failure modes is a statistically predictable time pattern of event production. In this paper we describe a novel approach for identifying faults that produce such a sequence of events, based only upon their temporal arrival pattern.

The approach uses state-of-the-art optimization techniques to establish a temporal similarity graph for a given group of alerts. Community detection algorithms are then applied to the event similarity graph in order to determine groups of faults with similar arrival patterns. We demonstrate the efficacy of the new approach by applying it to simulated event streams in realistic failure scenarios, reproducing results that are consistent with experience of deploying the technique in real world networks.

I. Introduction

The underlying hardware and software services running in an IT infrastructure generate huge quantities of operational status data. This data generally takes the form of a series of atomic, state-change notifications referred to as *events*. A typical enterprise IT infrastructure generates millions of events per day at rates of about 100 events per second. The largest enterprises generate orders of magnitude more event data.

Individual events do not always represent problems that require remedial action, application heartbeats, amongst many others, fall into this category. Within the millions of individual events generated each day, the typical large enterprise may have only a few hundred actionable incidents.

The process of identifying actionable incidents lies in the domain of *Fault Localization*. Fault localization encompasses a suite of different techniques that help isolate service-impacting failures and consequently any remediation actions that are required, techniques such as *Noise Reduction* and *Root Cause Analysis* (RCA). [1], [2] are excellent reviews of the field.

Historically, rules-based approaches have been used. Manual exclusion techniques such as blacklisting can be an effective form of noise reduction, [3], but they are difficult to maintain and are impractical at industrial scale. Root cause analysis systems such as [4], [5], [6], require behavioral models to be created, models that are intrinsically coupled to the infrastructure being managed. Not only are these models time-consuming to create, [7] but the tight coupling renders these techniques ineffective as soon as the underlying infrastructure changes. Moreover, and despite remaining key to some current commercial offerings, these RCA techniques are largely obsolete, especially given todays highly dynamic network and application infrastructures, [8], [9].

Artificial Intelligence (AI), specifically machine learning and data-driven techniques continue to gain traction and are now recognised alternatives to model based approaches [2], [10]. AI-based techniques have been applied across the fault localization domain. The high-level concepts behind data-driven noise reduction techniques are described in [11]. And by looking beyond the data held within an event and to the underlying network structure, [12] describes a method to identify those entities that are more likely to generate service-impacting incidents. An application of supervised machine-learning techniques for identifying root cause events is demonstrated in [13], while [14] shows that data-driven techniques have the capability to be more accurate and more resilient to change when compared with rules-based systems.

In this paper we explore further data-driven techniques to improve fault localization for a class of failure where the temporal nature of events is a key indicator of root cause.

We begin in Section II by describing our proposed technique, and introduce the necessary nomenclature in Sections II-A to II-C. Details of the different components of the algorithm are presented in Sections II-D to II-F, and we present the results of this analysis in Section III. The analysis was performed against simulated data, although our employer [15] has access to large amounts of real world data, this is covered by confidentiality restrictions. We close with Section IV where we discuss further enhancements and optimizations.

II. Architecture

A. Definitions and Nomenclature

The following definitions are used throughout this paper:

Event	An atomic notification from a network device, application or monitoring system. An event may not represent a fault, however a fault condition will result in at least one event being emitted.
Alert	A set of events grouped according to shared attributes. An alert will be closed once the underlying fault has been remediated.

978-3-903176-15-7 © 2019 IFIP

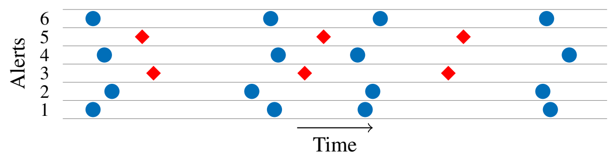

Fig. 1: Representation of event arrival times for multiple alerts

Incident	An actionable support item, such as a service interruption, requiring remediation.
Situation	An operational or causal grouping of alerts used to raise an Incident.

B. Motivation

There are countless different failure scenarios that can occur in the IT infrastructure of a large organization.

The majority of incidents are not created to describe catastrophic service outages. More common are incidents that encapsulate so-called “brown-outs”. For example, an individual service may experience increased database latency, causing increased response times for a product’s end-users. Such failures generally lead to events being generated asynchronously over a period of time as each entity becomes impacted.

A different class of failure, such as the complete failure of a piece of networking hardware or the failure of a specific process running on a server, tend to have a more immediate impact. In these cases, multiple events from different subsystems get generated at, or close-to, the same time, critically with a temporal pattern characteristic of alert propagation.

The aim of the current work is to automatically detect the second class of failure.

C. A Typical Use-Case

A schematic representation of the event arrival patterns for six different alerts is shown in Fig. 1. Each row represents an alert and each symbol within that row represents the arrival of an event. The alerts represented by the diamonds (alerts 1, 2, 4 and 6) have three, approximately co-occurring events. The alerts represented by the circles (alerts 3 and 5) have four, approximately co-occurring events. Visual inspection suggests a strong correlation between the arrival time of all of the events within each group.

The underlying premise of the proposed method is that a class of infrastructure failure exists where the failure of one entity will induce failure in connected nodes in a cascade. This manifests itself as a series of events being generated that are closely linked in time, have a temporal pattern that is repeated up to statistical variance, and sequential.

In the next sections we describe a novel technique that clusters alerts into distinct groups containing alerts with similar event-arrival patterns. Concretely, and for the case shown in Fig. 1, the technique would generate two clusters of alerts, or situations. One situation containing alerts 1, 2, 4, and 6 and a second situation containing alerts 3, and 5.

D. Similar Arrival Patterns

Techniques for event correlation, such as [12], [14], use the semantic and lexical similarity of alert attributes or the topological importance of a source device to correlate alerts. In this section we describe a technique to determine a similarity measure for two alerts based purely upon event arrival-time.

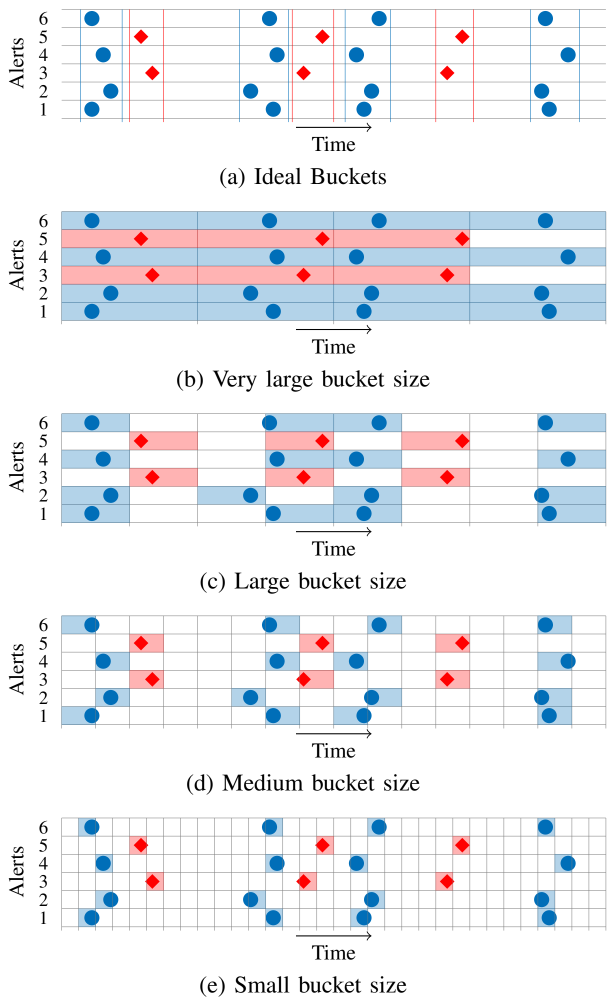

Fig. 2: Representation of different bucketization schemes

Our approach to determining alert similarity is based upon time-bucketing. We divide time into equal sized buckets and assign each event to a bucket based upon its arrival time.

Fig. 2 shows the concept of time bucketization and qualitatively, how bucket size can impact upon accuracy. In this example the alerts should be grouped into two distinct clusters; one containing alerts 1, 2, 4 and 6, the second alerts 3 and 5. An idealized discretization scheme is shown in Fig. 2a.

Fig. 2b shows how events get bucketized when a very large bucket size is used. In this case almost all alerts occur in the same buckets, and it is likely that all alerts would be considered similar. As the bucket size is reduced, Fig. 2b to Fig. 2e, alerts that should be considered similar, fall into different time buckets. The events of alert 1, 2, 4, and 6 in the first column of Fig. 2c get split across multiple buckets in Fig. 2d. Similarly the events from alert 3 and 5 in the third column of Fig. 2d get split across two columns in Fig. 2e.

Intuitively, a large bucket size leads to an increase in the number of false positives i.e. alerts being considered similar when they are not, whereas a small bucket size leads to a

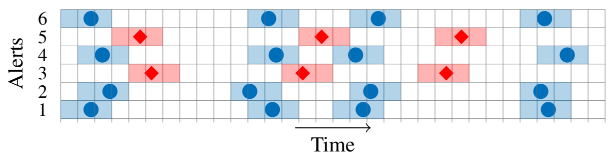

Fig. 3: Bucketization with small buckets and arrival spread

larger number of false negatives, i.e. alerts being considered dis-similar when they are in fact correlated.

In addition to bucket size, the temporal position of an alert in a bucket also impacts accuracy. Two events may arrive within a fraction of a second, but, depending upon the bucket-boundary location, those events may end up in different buckets, again leading to an increase in false positives.

In order to overcome these limitations the approach taken in the current study is to use a small bucket size but to "spread" the event arrival across multiple buckets. A schematic representation of this approach is shown in Fig. 3.

Once each event has been assigned to the appropriate buckets, the alert similarity can then be calculated using any of the standard set similarity algorithms.

We begin the description of the algorithm with some formal notation. A set of events, E is collected over time T. These events are themselves effectively a feature vector, containing attributes such as the source of the event, time of arrival and so on. We can formally define an equivalence relation between events, where two events $E_i \sim E_j$ if there is a similarity function $s(E_i)$ definable on the attributes of the event such that $s(E_i) = s(E_j)$. Practically, such an equivalence relationship would, for example, conclude that all "Ping Fail" events for the same host are equivalent to each other and can be represented by a single alert, a process known as de-duplication. In this way we can define the set of alerts $A = \{A_i\}$ as the set of equivalence classes of E as follows:

$$A_i = E/\sim \tag{1}$$

$$E = \textstyle\bigcup_i A_i \tag{2}$$

$$A_i \cap A_j = \varnothing\,. \tag{3}$$

Time is split into N_b equally-sized time-buckets of duration δT_b. An event, arriving at time t is assumed to have arrived across N_s time-buckets centred on t.

We construct an alert feature matrix, $\mathbf{F}_{|A|\times N_b}$ such that $\mathbf{F}_{ij} = 1$ when an event from alert i has arrived in time-bucket j, and is elsewhere zero. We can then construct a similarity matrix $\mathbf{M}_{|A|\times|A|}$, as $\mathbf{M}_{ij} = Sim(F_i, F_j)$ where $Sim()$ represents a function to calculate an arbitrary similarity measure.

In the current contribution we use the Jaccard similarity, $\mathbf{J}$, where:

$$\mathbf{J}(X_1, X_2) = \frac{|X_1 \cap X_2|}{|X_1 \cup X_2|}\,, \tag{4}$$

and X_1 and X_2 represent arbitrary sets. In the current case

$$\mathbf{M}_{ij} = \frac{|f_i \cap f_j|}{|f_i \cup f_j|}\,, \tag{5}$$

where $f_i = \{k : F_{ik} = 1\}$.

Calculating $\mathbf{M}_{ij}$ requires a pairwise comparison of every element in A, a so-called *Similarity Join*. A naive implementation scales as $O(|A|^2)$, a prohibitively expensive operation for a set with large cardinality and when results are required in real time. To reduce the time complexity of the calculation there are optimization techniques such as prefix filtering, [16], [17]. For the current study we use the approach in [18] but adapted for the Jaccard similarity measure.

A key part in all the optimization techniques is the concept of a similarity-threshold, θ_{sim}, below which, two elements are considered to be completely dis-similar. In our case we define $\mathbf{M}^\theta$ by zeroing all elements of $\mathbf{M}$, when $\mathbf{M}_{ij} \leq \theta_{sim}$.

While the primary motivation for θ_{sim} is computational efficiency, it has practical utility in our case. Intuitively, there is a similarity below which it makes little sense for us to correlate alerts. For example, $\theta_{sim} = 0.5$, may be an obvious choice, it is the point at which there are more uncorrelated arrivals than correlated arrivals. In our case we are only interested in alerts, whose arrival patterns show high degrees of similarity. The analyses presented in Section III use $\theta_{sim} = 0.8$.

E. Graph Theory & Community Detection

One of the main drawbacks in the application of unsupervised clustering techniques such as k-means and non-negative matrix factorization, is the requirement for k, the number of clusters, to be pre-defined. The optimum value for k depends upon the data being analysed and while techniques such as the silhouette score, [19], can be used to optimize k, they are interactive techniques that do not lend themselves to an automated process. Ultimately, and regardless of whether a partitioning approach such as k-means or a density-based clustering technique is chosen, a parameter needs to be pre-defined that dictates how the data will be grouped. One of the primary motivators for our choice of clustering technique is to eliminate the need for such a parameter.

In this and subsequent analyses we use standard graph theory notation, and make use of the adjacency matrix $\mathbf{B}$, where $\mathbf{B}_{ij} \neq 0$ represents the links between nodes i and j, additionally, we consider only simple undirected graphs.

For the set of alerts, A, the output of the method described in II-D is an alert similarity matrix, $\mathbf{M}^\theta_{|A|\times|A|}$. The properties of $\mathbf{M}^\theta$, are identical to those of a edge-weighted adjacency matrix and hence any analysis techniques applicable to $\mathbf{B}$ can be equally applied to $\mathbf{M}^\theta$.

Community detection algorithms aim to group the vertices of a graph into distinct sets, or communities, such that there exist dense connections within a community and sparse connections between communities. Intuitively a community detection algorithm applied to $\mathbf{M}^\theta$, will generate a set of alert groupings, I, where each group contains a set of alerts whose events share similar arrival patterns.

A review of the different community detection algorithms is available in [20]. The criteria against which any algorithm must be assessed in order to gain adoption within the current context, are accuracy, scalability, memory consumption, and speed. For the purposes of the current analysis we concentrated only on community-detection algorithms based upon modularity maximization, specifically the well-known *Louvain* algorithm, [21].

F. Situation Creation

The final step in our approach is the conversion of the communities, I, into the set of situations, S. To create them we introduce a new free parameter of the model θ_{alerts}, which is the minimum cardinality of any discovered community $I_i \in I$. We can then define the situations discovered as:

$$S = \{S_i\} \tag{6}$$

$$S_i \subset A \tag{7}$$

$$S_i \cap S_j = \varnothing \tag{8}$$

$$S = \left\{ I_i \in I \,\middle|\, |I_i| > \theta_{alerts} \right\} \tag{9}$$

$$\neg\Box\ (A = \textstyle\bigcup_i S_i)\,. \tag{10}$$

The properties of S, defined by equations 8, 9 and 10, have important operational significance: an alert must appear in at most one situation; a situation should contain at least θ_{alerts} alerts; and an alert is not required to appear in a Situation.

III. Analysis

In this section we quantify the efficacy of the technique from Section II against a subset of the failure scenarios seen in IT infrastructures.

Owing to a lack of ground-truth event data, we have simulated several failure scenarios. Each of the failure scenarios outlined below was overlaid on a randomly generated background event-stream and the algorithm of Section II executed against it. Approximately 20 minutes of background event data was generated at an average rate of 100 events per second. The inter-event arrival times were generated to approximate a distribution observed in a real environment.

A. Failure Scenarios

Out of all the failure scenarios observed in an IT infrastructure only a subset exhibit the temporal patterns of alert propagation for which the current detection method has been devised. For this study we focus on four such scenarios.

1) ***Flapping Interface*****:** A common failure mode for interfaces on switches or routers is for a port to repeatedly swap between "up" and "down" states in rapid succession. On interface failure, connected devices generally produce, near instantaneously, an asynchronous accessibility error such as a *LinkUp* or *LinkDown* trap. These failures can be of arbitrary duration and are characterized by state-change events occurring at a rate of one every few seconds up to multiple per second.
2) ***Container Failure*****:** Representative of the failure of a "Container" and its contained entities such as a virtual machine (VM) server and the VMs it manages. When a VM server fails, the management system will, in general, generate an event to indicate the server failure followed by separate events to notify the inaccessibility of each contained VM. An individual server may contain many tens of VMs. Notification of a VM failure generally occurs within 30 seconds of the server failure.
3) ***Service Failure*****:** The failure of an IT service can often be characterized by the events generated by dependant applications. We characterize this failure by distinct bursts of events generated over short time-span of up to a few seconds. Each burst occurs at random intervals as different applications encounter the same fault.
4) ***Intelligent Polling*****:** Following the failure of an interface or routing device, a common management technique is to periodically poll the devices that have become disconnected until they become accessible. Each inaccessible device would be polled on a round-robin basis at rate of approximately once per minute. Many hundreds of devices may be impacted in this way. The duration of each polling sweep will depend upon the number of impacted devices, but is usually of the order of seconds.

TABLE I: Failure Scenario Parameters

	Failure Duration (secs)	Alert Count	Burst Duration (secs)	Inter-Burst Time (secs)
Flapping Interface	60, 300, 600, 1200	12, 24, 48	-	1, 1-5, 2-10
Container Failure 1	-	12, 24, 48	1	-
Container Failure 2	-	20, 30, 50	30	-
Service Failure	60, 300, 600, 1200	10, 20, 30, 50	5	5-30
Intelligent Polling 1	300, 600, 1200	100	5	60
Intelligent Polling 2	300, 600, 1200	200	10	60
Intelligent Polling 3	300, 600, 1200	500	30	60

Each of the above failure scenarios can be characterized by some, or all, of the following parameters

Failure Duration	How long the failure persists for.
Alert Count	Number of alerts each failure generates.
Burst Duration	Time over which a burst of events occurs.
Inter-Burst Time	Time between event bursts.

Table I shows the different parameters used to define each failure scenario. Note that in the Flapping Interface scenario, there is no burst duration as all the events are created instantaneously. Similarly, for a Container Failure, failure duration and the inter-burst time do not apply.

B. Results

In addition to the parametric definitions of each failure scenario, Section II outlined several parameters that control behavior of our algorithm. To limit the number of experimental permutations we fixed several parameters across all experiments as shown in Table II.

We have analyzed the results using well known measures of algorithmic effectiveness, but the nature of the experiment calls for some particular choices regarding those metrics. Alongside the well-known F_1 score, we also make use of the *False Positive Rate* (FPR) and the specific values for incorrectly clustered alerts and incorrectly observed situations.

TABLE II: Algorithm Parameters

Parameter	Value
Event Collection Period, T (secs)	1200
Similarity Threshold, θ_{sim}	0.8
Alert Threshold, θ_{alert}	4
Bucket Size, δT_b (secs)	5
Arrival Spread, δT_s (secs)	15, 30, 45, 60

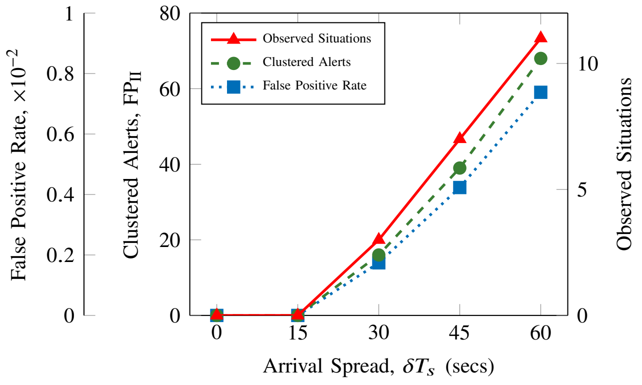

Fig. 4: Accuracy Metrics for the Background Event Stream

To calculate the F_1 score and FPR, we adopt the following definitions:

True Positive	A failure alert correctly assigned to a situation. If the failure alerts are assigned to multiple situations, the largest is used.
False Positive I	A failure alert correctly included in a situation but where a larger proportion of failure alerts are in a larger situation.
False Positive II	An alert from the background event stream included in a situation.
True Negative	An alert from the background event stream not included in a situation.
False Negative	A failure alert not assigned to a situation.

The sum of *False Positive I* (FP_I) and *False Positive II* (FP_{II}) is used to calculate the F_1 score.

An integral part of our analysis is the randomly generated, background event field. The objective of any unsupervised machine learning technique is to discover hidden patterns within a given dataset. To identify any co-incidental patterns, we executed our method against the background field. Fig. 4 shows the accuracy metrics for the analysis, specifically: the FPR, FP_{II}, and the number of *Observed Situations*, OS. Ideally, the value of each metric would be 0 for all values of the *Arrival Spread*, δT_s. In this analysis the qualitative trends of FP_{II} and OS are more important than specific values. We observe that for $0 \leq \delta T_s \leq 15$, no situations are created. As δT_s increases, the number of co-incidentally observed situations rises, a relationship, that whilst undesirable, is expected. At $\delta T_s = 60$, 11 situations are observed grouping a total of 68 alerts, with an FPR of only 0.74×10^{-3}. To ensure a consistent assessment of our technique, these "co-incidental" situations have been subtracted from the results of all subsequent experiments.

A high-level assessment of our technique, across all failure scenarios and for all permutations of the algorithm parameters,

TABLE III: Number of Experiments with F_1 = 1.0

Failure Scenario	Failure Duration (secs)					Expt. Count
	60	300	600	1200	All	
Flapping Interface	36	36	36	36	144	144
Container Failure 1	-	-	-	-	12	12
Container Failure 2	-	-	-	-	0	12
Service Failure	14	15	15	15	59	64
Intelligent Polling 1	-	2	2	2	6	12
Intelligent Polling 2	-	1	1	1	3	12
Intelligent Polling 3	-	1	1	1	3	12
Total	50	54	54	54	221	268

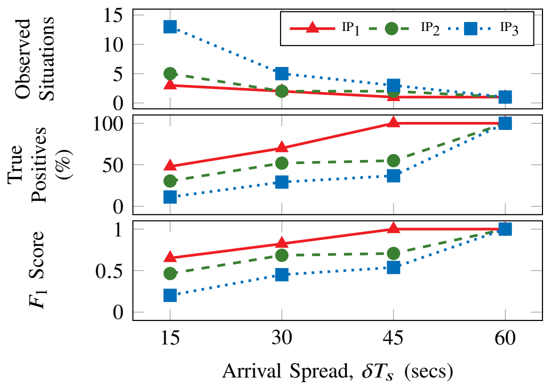

Fig. 5: Accuracy Metrics for the Intelligent Polling Scenarios

is shown in Table III, specifically the number of experiments where F_1 = 1.0 was obtained. Perfect precision and recall were achieved in 221 out of a total of 268 experiments. For two failure scenarios, the *Flapping Interface* (FI) and *Container Failure 1* (CF_1) cases, no errors were observed and for the *Service Failure* (SF) case over 92% of the experiments contained no errors. The main reason for the accuracy of the FI, CF_1 and SF failure cases is the short burst duration of 5 seconds or less. We also note that the duration of each failure had little impact on the accuracy of the method.

Further analysis revealed there to be only 5 cases where FN > 0, and no cases where $FP_{II} > 0$. Directly put, the only alerts that were ever clustered into situations were associated with a failure and in 263 of the 268 experiments *all* of the failure alerts were associated with an actionable situation.

We now examine the results from the *Intelligent Polling* ($IP_{1\text{-}3}$), CF_2, and SF failure scenarios in more detail.

For the SF scenario, we observed only 5 cases (out of 64) where we didn't achieve an F_1 score of 1.0. All those cases used δT_s = 15, and all the failure alerts were correctly identified (i.e. FN = 0). The inaccuracy in the F_1 score occurs because the failure alerts were grouped into two situations rather than one. For values of $\delta T_s \geq 30$ all the failure alerts were correctly grouped into a single situation.

Because the duration of a failure has little impact on accuracy, the results presented for the IP failure scenarios have been averaged over all the values of failure duration for each value of δT_s. The results are shown in Fig. 5, in addition, for all cases, our method gave FN = 0. The results show a clear relationship between δT_s and accuracy. As δT_s increases the

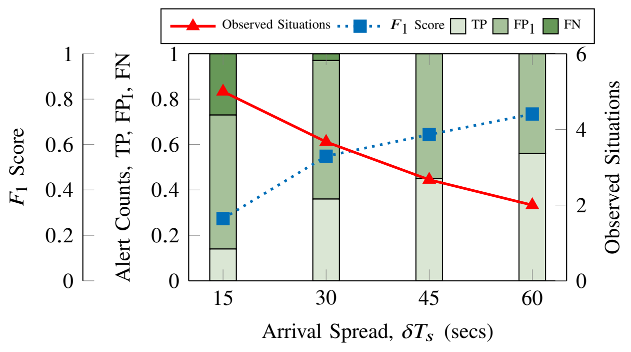

Fig. 6: Accuracy Metrics for the CF_2 Scenario

F_1 score approaches 1.0, the number of situations reduces to the expected value of 1 and the proportion of failure alerts clustered into that single situation increases to 100%.

At each value of δT_s the accuracy of the method improves as the burst duration reduces. For $\delta T_s = 15$ the number of situations drops from 13 to 3, for burst duration of 30 and 5 seconds respectively. The larger the value of δT_s in relation to the burst duration, the more likely that alerts occurring at different times can be considered co-incident and consequently posses a high temporal similarity. For a given set of alerts, the higher their similarity, the larger each community will be and hence fewer situations will be generated.

The results for CF_2 are shown in Fig. 6. The main variable in this case was the number of alerts generated by each failure, which was observed to have little effect on the results. For each δT_s we present average values for the F_1 score, OS, and the counts of each failure alert classified as TN, FP_I, or FN, expressed as a proportion of all failure alerts. The qualitative behaviour of our method against CF_2 is inline with other scenarios, however it is the only one where the method failed to achieve an F_1 score of 1.0 and where FN > 0.

The reduced performance for the CF_2 case, when compared with IP_3, which uses the same value for the burst duration, is that CF_2 has only a single event per alert. For the IP cases each alert contains repeating events, increasing the likelihood of alerts sharing co-occurring events. Comparison of CF_2 with CF_1, shows directly the impact of burst duration on accuracy.

Much of our analysis has focussed on those failure scenarios where our method didn't achieve perfect results. While analysing the reasons for inaccuracies is important, it is also important to view the discussion in the context of the method's generally high accuracy, as shown in Table III.

IV. Conclusions

In this work we have described a novel algorithm for fault localization. The algorithm exploits the categorical equivalence between matrices and graphs, to transform the clustering of temporal similarity into the well understood exercise of modularity detection in graphs. Applied to realistic real world scenarios, we demonstrate that it is effective in capturing and clustering together causally related alerts resultant from failures that posses a temporal pattern of alert propagation. This success persists even when the alerts are embedded in a random background. A variant of this algorithm is in commercial use at a number of enterprise scale networks, and is the subject of a patent filing by our employer Moogsoft.

There are a number of interesting future avenues for further research to extend and investigate this and related approaches. In particular the question of random background removal is an interesting one, and, continuing to improve the performance of the approach in high noise environments is a particularly important problem. Similarly, the ability to cope with failures of radically different cascade parameters will further enhance the utility of the approach. More theoretically, the algorithm used one particular definition of graph modularity, out of many alternatives. It is an open question if other techniques of module detection will produce different results. Indeed, the central insight that led to this algorithm (the close relationship between matrices and graphs) is also, we believe, a rich area for the development of further fault localization approaches.

References

[1] M. ł. Steinder and A. S. Sethi, "A Survey of Fault Localization Techniques in Comput. Networks," *Sci. of Comput. Programming*, 2004.

[2] A. Dusia and A. S. Sethi, "Recent Advances in Fault Localization in Comput. Networks," *IEEE Communications Surveys & Tutorials*, 2016.

[3] L. Metcalf and J. M. Spring, "Blacklist Ecosystem Analysis Spanning Jan 2012 to Jun 2014," *ACM Digital Library*, 2014.

[4] "IBM Tivoli Netcool/OMNIbus," 2017. [Online]. Available: http://www-03.ibm.com/software/products/en/ibmtivolinetcoolomnibus

[5] "EMC Automated Data Center Manager," 2017. [Online]. Available: http://www.emc.uz/it-management/smarts/index.htm

[6] "NetExpert Datasheet," 2016. [Online]. Available: http://www.mycom-osi.com/products/netexpert-fault-service-impact-management

[7] S. Kliger, S. Yemini, and Y. Yemini, "A Coding Approach to Event Correlation," *Network Management IV*, 1995.

[8] M. Miyazawa and K. Nishimura, "Scalable Root Cause Analysis Assisted by Classified Alarm Information Model Based Algorithm," in *7th Int. Conf. on Network and Service Management*, 2011.

[9] Smarts, "Downstrean Suppression is Not Root Cause Analysis," Tech. Rep., 2002.

[10] P. Bodík, "Automating Datacenter Operations Using Machine Learning," Ph.D. dissertation, 2010.

[11] R. Harper, "Entropy & The Science of Noise," 2016. [Online]. Available: https://www.moogsoft.com/whats-new/entropy-noise/

[12] P. Tee, G. Parisis, and I. Wakeman, "Vertex Entropy as a Critical Node Measure in Network Monitoring," *IEEE Transactions on Network and Service Management*, 2017.

[13] R. Harper and P. Tee, "The Application of Neural Networks to Predicting the Root Cause of Service Failures," in *IFIP/IEEE Symp. on Integrated Network and Service Management*, 2017.

[14] ——, "Cookbook, a Recipe for Fault Localization," in *IFIP/IEEE Network Operations and Management Symp.*, 2018.

[15] Moogsoft, "Moogsoft Documentation." [Online]. Available: https://docs.moogsoft.com/

[16] W. Wang, "Similarity Join Algorithms: An Introduction," Tech. Rep., 2008. [Online]. Available: https://www.cse.unsw.edu.au/~weiw/project/tutorial-simjoin-SEBD08.pdf

[17] S. Chaudhuri, V. Ganti, and R. Kaushik, "A Primitive Operator for Similarity Joins in Data Cleaning," in *22nd Int. Conf. Data Eng.*, 2006.

[18] R. J. Bayardo, Y. Ma, and R. Srikant, "Scaling Up All Pairs Similarity Search," in *Proc. of the 16th Int. Conf. on World Wide Web*, 2007.

[19] P. J. Rousseeuw and Peter, "Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis," *Journal of Computational and Applied Mathematics*, 1987.

[20] B. S. Khan and M. A. Niazi, "Network Community Detection: A Review and Visual Survey," 2017. [Online]. Available: http://arxiv.org/abs/1708.00977

[21] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast Unfolding of Communities in Large Networks," *Journal of Statistical Mechanics: Theory and Experiment*, 2008.