An unsupervised approach to discover filtering rules from diagnostic logs

Marcello Cinque[†⋆], Raffaele Della Corte[†‡], Giorgio Farina[†⋆], Stefano Rosiello[‡]
[†]*DIETI - Università degli Studi di Napoli Federico II, via Claudio 21, 80125 Naples, Italy*
[⋆]*CINI - Consorzio Interuniversitario Nazionale per l'Informatica, M.S. Angelo, Via Cinthia, 80126 Naples, Italy*
[‡]*Critiware s.r.l., via Carlo Poerio 89/A, 80121, Naples, Italy*
{macinque, raffaele.dellacorte2, giorgio.farina}@unina.it, stefano.rosiello@critiware.com

***Abstract*—Diagnostic logs represent the main source of information about the system runtime. However, the presence of faults typically leads to multiple errors propagating within system components, which requires analysts to dig into cascading messages for root cause analysis. This is exacerbated in complex systems, such as railway systems, composed by several devices generating high amount of logs. Filtering allows dealing with large data volumes, leading practitioners to focus on interesting events, i.e., events that should be further investigated by analysts.**

This paper proposes an unsupervised approach to discover filtering rules from diagnostic logs. The approach automatically infers potential events correlations, representing them as fault-trees enriched with scores. Trees define filtering rules highlighting the interesting events, while scores allow prioritizing their analysis. The approach has been applied in a preliminary railway case study, which encompasses more than 710k events generated by on-board train equipment during operation.

***Index Terms*—Filtering, Fault-tree, Data analysis, Event logs**

I. Introduction

Diagnostic logs are the primary source of information for understanding the runtime behavior of a system. Diagnostic logs are sequences of text lines –typically stored in log files– reporting on the runtime behavior of a system [1], including entries highlighting the occurrence of system failures. Their analysis has been extensively used for troubleshooting [2]–[4]. However, leveraging diagnostic logs for root cause analysis, i.e., identifying the fault originating failures occurred during system operation [5], is a challenging task. Human experts generally rely on their experience to manually identify anomalous log entries, often by querying for predefined keywords. Understanding and traversing the diagnostic data logs from different components demands for substantial cognitive work by human experts. Highly specialized analysts are expected to face a number of challenges, which encompass the high volume and heterogeneity of data, the presence of different error and failure mode, the presence of corrupted, duplicated or redundant data, or even worst, the absence of data to infer the root cause of the occurred problem, the absence of consolidated and automatic analysis procedures. This is exacerbated in complex systems like Train Control and Monitoring System (TCMS) [6], which include a wide set of heterogeneous subsystems, each one generating diagnostic logs.

Filtering is one of the analysis techniques allowing to deal with large data volumes by supporting the identification of interesting events [7], i.e., events that should be followed up by analysts for further investigation because they may help revealing root cause of occurring failure. *Filtering rules* allow retaining events that are expected to have been generated under exceptional conditions, e.g., failures, and discarding the events generated by normative operations or containing redundant information with respect to the retained ones. The discovery of filtering rules in an automatic way represents a valuable tool for domain experts, since it can suggest potential events correlation across huge data volumes, which can be reviewed by the experts for validation. In this respect, different studies propose methods to infer events relationship from logs, representing them as fault-tree [8]–[10]. However, existing approaches require labeled datasets, e.g., datasets where events are already grouped together and the occurrence or not of the interesting event is provided for each group, which can be difficult to obtain for real-world production systems.

In this paper we propose an unsupervised approach to automatically discover filtering rules from diagnostic logs. The approach leverages both temporal coalescence and unsupervised machine learning techniques to automatically infer potential events correlations from diagnostic data, and extract filtering rules. The inferred rules are represented as *fault-tree* models, in order to provide to domain experts a comprehensive and simplified representation for review. A novel scoring mechanism is also proposed to provide each rule with a numerical score, which allows experts to prioritize the validation of the obtained rules as well as to easily discard potential wrong filtering rules. We applied the proposed approach to a preliminary railway case study in the context of the AID4TRAIN project, which is being developed by Critiware S.r.l., an industrial railway partner acting as problem owner and data provider (Hitachi Rail Italy), and with the support of a national research center (CINI). The case study encompasses diagnostic logs collected through the TCMS of a high-speed train. The dataset includes more than 710k events collected in a production environment during eight weeks of train operation.

The paper is organized as follows. Section II introduces a motivating example for our proposal. Section III describes our approach, encompassing both event-tree learning and scoring mechanisms. Section IV presents the results in our railway case study. Section V discusses both background concepts and related work, while Section VI concludes the work.

II. Motivating example

Diagnostic equipment available on modern systems are able to automatically report information related to potential faults or functional anomalies. However, as anticipated, the consequences of a fault are hardly confined on the component that is first affected by the problem, and can have cascade effects (and related diagnostic messages) on many components. This leads to log with an excessive number of lines, which hardens the root cause analysis task, impacting to the overall life-cycle cost of a working system, due to the need of highly specialized maintenance staff. The analysis is impaired by several factors: data volume and heterogeneity, variety of fault types, presence of corrupted, redundant or repeated data, and, in rare cases, lack of useful information to reconstruct the event of interest.

While there is still the need for human experts, to ultimately judge the hypothesized cause of a fault and program a maintenance action, the problem we aim to solve is to reduce the gap between the expert and the raw data, by providing a simplified view of knowledge automatically extracted from data in the form of fault trees. The challenge we address in this paper is to infer these models by mining potential correlations across huge volumes of data, spanning significant hours of functioning of the on-board equipment, in production environments. Learned grouping rules can be then used, after passing the expert review, to automatically filter previously unseen raw data, hence reducing the amount of information to be delivered to the specialist, in case of anomalies.

As an example, a sequence of events happening multiple times in the logs of our railway case study is related to the emergency braking system of a train, which is depicted in Fig. 1. When an emergency brake is activated, the event is reported by multiple components of the train (such as the braking system, the vehicle, and monitoring and diagnostics system). These events happen near in time and are strictly correlated. Our approach aims to capture this correlation by analyzing historical diagnostic logs collected during the system runtime, and propose a simplified view to the maintenance experts by highlighting the primary event (i.e., `Emergency brake activated in A48`) and marking as secondary the remaining ones as they can be easily filtered out. However, sequences of recurring events not always indicate a potential filtering rule. Indeed, in Fig. 2 is shown an example of a sequence reporting events due to multiple persistent failures (i.e., in the WC and in the HVAC systems). These three events are not due to a common root cause but since they are persistent, they are reported multiple times by diagnostics close in time. Multiple concurrent faults can mislead the automatic learning of filtering rules. Our approach propose a scoring mechanism (described in Section III-B) in order to spot those cases and discarding potential wrong filtering rules, therefore reducing the effort of the domain expert in verify and validate them.

```
…
Code 81 : _BREMG_A48 «Emergency brake enabled in A48»
Code 383: _VEIC_P03  «Received manual braking command»
Code 410: _MDS_P18   «Emergency brake»
…
```

Fig. 1. Example of recurring events caused by cascading effects.

```
…
Code 14 : _HVAC_CAB1   «Loss of communication with HVAC cabin 01»
Code 710: _FltToiletT1 «WC out of order in sector T1»
Code 810: _PressureFlt «Fault in tank pressure switch WC sector T1»
…
```

Fig. 2. Example of recurring events caused by different concurrent faults.

III. Proposed approach

The proposed approach aims to support maintenance processes by providing practitioners with potential filtering rules. Each rule highlights the event to consider as *primary*, i.e., the interesting event that should be retained, and the ones to consider as *secondary*, i.e., the events that can be discarded since they can be considered as consequence of the primary event. Moreover, each rule is characterized by a score summarizing the correlations between events composing the rule, which provides an immediate feedback to domain experts about the validity of the rule, as well as a prioritization for the rules validation. The approach leverages two main stages: (i) *Rule discovery*, which aims to infer filtering rules from diagnostic log in an automatic way, representing them as fault-tree; (ii) *Scoring*, which enriches the obtained rules with the score. Fig. 3 depicts the steps composing the proposed approach. We first describe the steps of the Rule discovery stage, i.e., Tupling, Clustering and Fault-Tree building, and then the Scoring stage.

A. Rule discovery

The Rule discovery stage leverages both temporal coalescence and unsupervised Machine Learning (ML) techniques to infer potential filtering rules, and represent them as fault-tree. This stage encompasses the following three steps.

Tupling. The event-tupling is a widely recognized technique [11] for temporal coalescence. A tuple is a set of events which are close in time; the assumption here is that often multiple events that are reported together are due to the same underlying anomaly. Moreover, the same anomaly may persist, repeats often over time, and propagates to other components, which in turn may report an additional event into the log. Two subsequent events are included in the same tuple if the time elapsed between their occurrences is less than a fixed time window. Otherwise, they will be grouped in different tuples. In this process the choice of the window size is a crucial factor, and represents a configuration of our approach. The obtained tuples are then represented as a binary sequence, with each element indicating the presence of an event in the tuple.

Clustering. In this step, the tuples encompassing a similar set of events are grouped together in a cluster leveraging an unsupervised ML technique, in order to discover potential filtering rule. To this aim the step makes use of a hierarchical clustering technique based on single linkage [12]. To measure the distance between two tuples, the step uses the hamming

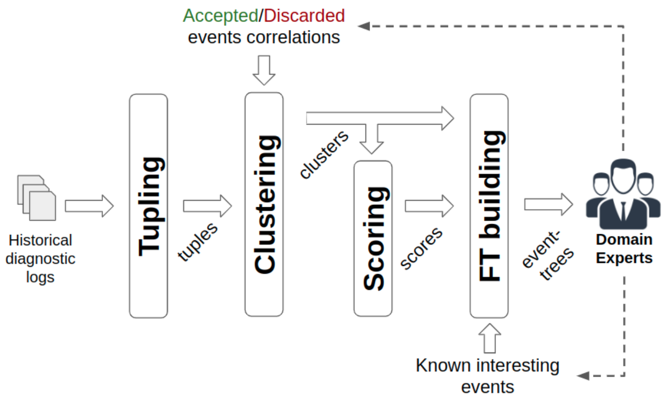

Fig. 3. Overview of the proposed approach.

distance, which is directly proportional to the number of different events occurred in the tuples. For each cluster, the number of events composing the clusters, the probability of occurrence of the event within the cluster, and the average duration of the tuples of the cluster are computed. Each cluster is considered to be a potential learned filtering rule if it contains at least two events and at least two tuple. Indeed, clusters with a single tuple are likely due to chance, while clusters with less than two events are not useful to learn potential correlations between events.

Fault Tree building. Each cluster is transformed in a two-level fault tree. The root of the tree is the *primary event*, which is selected depending on the list of *known interesting events* (provided as input), i.e., events that domain experts already known to be relevant in the target system (such as events representing potential root cause of failures). If the cluster contains an interesting event indicated in the list, this event is used as primary event of the tree; otherwise, the primary event is represented by a potential unknown event, which needs to be investigated from domain experts. The remaining events of the cluster are used as leaves of the tree, indicating *secondary events*. Each tree represent a filtering rule, which can be summarized as follows: when all the events composing the tree occur within the related time window (i.e., the average duration of the tuples within the cluster), the root event of the tree is marked as primary, while all other events as secondary. This allows analysts to easily filter-out all the secondary events from logs, focusing only on the primary ones when analyzing diagnostic data for root cause analysis.

The domain experts are expected to review the obtained fault trees in order to consolidate the related rule. To this aim each rule is translated in a XML standard representation[1] of a fault tree, which can further elaborated, discarded and accepted by a domain expert by means of a graphical fault-tree editor. Moreover, the *score* provided by the scoring step (described later) is attached to each fault tree to support their analysis. It is important to note that the reviews made by experts are provided as inputs to the clustering step, which accepts the list of *accepted* and *discarded* rules. This prevents that rules already analyzed by experts will be reviewed again. Further, if the experts identify new primary events during the analysis, they will be included within the *known interesting events* list.

B. Scoring

The score reduces the manual effort of domain experts allowing both cluster filtering and prioritization. In details, the score helps the domain experts to (i) test the validity of clusters and (ii) assign a priority to clusters in order to guide their validation. This is an important feature since, despite the tupling tuning obtained from the sensitivity analysis works well in the average cases, there are corner cases that need to be detected and handled. Timing coalescence might group independent event sequences within the same tuple when the frequency of these events together is not negligible (collision). At the same time, it might split an independent diagnostic event sequence in different tuples when the time window is too tiny (truncation). Our scoring mechanism mines the dataset to check for these phenomena, and provides a score describing the correlations between the cluster events. While the clusters classified as collisions or truncation are discarded due to lack of empirical evidence, the remaining clusters are provided to the expert domains as several DAGs (Directed Acyclic Graphs), where each node represent a cluster. DAGs show the inclusivity relationships between clusters as well as the score of the clusters. This representation helps the expert domains to guide the validation, and enables the score computation.

Each DAG is analyzed from the leaves. For simplicity, we describe one iteration of DAG exploration to show how our score is calculated. For each leaf, the score is calculated considering all binary combinations of the events composing the related cluster. In details, for each binary combination of events E_i, E_j, we calculate a correlation Index in both direction, i.e., $E_i->E_j$ and $E_j->E_i$, according to the following equation:

$$Index_{E_i->E_j} = \frac{P(E_j|E_i) - P(E_j)}{1 - P(E_j)} \tag{1}$$

Then a score is calculated for the events couple, as the maximum between the two calculated index:

$$Score_{E_i,E_j} = max\{Index_{E_i->E_j}, Index_{E_j->E_i}\} \tag{2}$$

In order to understand the proposed correlation index, let us give this illustration: considering $Index_{E_i->E_j}$, the event E_i affects E_j only if the frequency of E_j is higher when E_i occurs. Hence, the Index subtracts the frequency of E_j when E_i is verified with the frequency of E_j in the entire dataset. The obtained Score has two operational ranges of values: (i) score less than or equal to zero, indicating that E_i and E_j could be independent events that are together for a chance; (ii) score from zero to one, which estimates the maximum achievable correlation between the two events. We choose the maximum in (2), because (for instance) if we have a diagnostic event that generates a fault, the presence of that event does not always imply a fault, but it generally causes the fault. In that case, the score is close to one because it chooses the maximum correlation between the two directions. Once we estimate the score for each binary combination, the score of the cluster is calculated as the arithmetic means of the event binary combinations:

$$Score_{C_i} = \frac{\sum_{i=1}^{N-1}\sum_{j=i+1}^{N} Score(E_i, E_j)}{\frac{N*(N-1)}{2}} \tag{3}$$

If this score is negative, we can classify that cluster as an aggregation of independent events (i.e., a collision). While if the score is positive, it is adopted as a priority metric to guide expert validation. Subsequently, we can build the score of the parent cluster from the leaves. Since the the parent cluster includes all the events of its children, we computed the parent score by averaging solely the new binary combinations that are not included in the children. This allows evaluating only the added value of the parent cluster with respect to its children. Again, if the score is negative, we can conclude that the events added from the parent cluster have no significant effect and thereby the cluster can be classified as a collision. A particular case is when the new cluster does not add new combinations of events, we can classify that case as evidence

[1]The Open-PSA Model Exchange Format is used as XML representation.

of tupling truncation, keeping the parent while classifying the direct children as truncations.

In conclusion, the clusters classified as collisions or truncations are not provided to the expert domains for lack of empirical evidence. In contrast, the clusters with positive scores are provided in a DAG with the associated scores (priority) to guide the expert validation from the cluster, which includes more probably rules.

IV. Railway case study

As a preliminary case-study, we collected eight weeks of diagnostic logs generated by the TCMS of a single high-speed train during operation; the collection has been made by means of the train control room facility at our industrial partner. The collected dataset is not labeled; therefore, no information is provided about the relationship of the events. Before analyzing the dataset with our approach, which has been implemented as Python-based modules, we execute a **data preparation** step. This step aims to transform the data in the format expected by the proposalas well as to address suggestions provided by our industrial partner. First, the raw output of the diagnostic systems is converted in a table, containing for each event: a timestamp, the event-type and the event-code (which are described in the train model of the specific train), a description in natural language of the event and the name of the component affected. The obtained table is then analyzed to discard duplicated events (i.e., events with the same event-code and timestamp), according to industrial partner indications.

A. Rule discovery statistics

The dataset obtained after the data preparation step is composed by 716,062 events, observed in a period of eight weeks. During this time-frame 1,364 unique events are observed. In order to properly configure our proposal, we first performed a sensitivity analysis in order to select the tupling window. Fig. 4 shows the number of tuples by the selected tupling window. We selected the window size that correspond to the knee point of the curve, which represents a good tradeoff according to the previous study in [11]. Therefore, we considered a tupling window of 5 seconds, corresponding to 53,297 tuples.

Configured the tupling window, we run the proposed approach on the prepared dataset. After the execution of the clustering step, we obtained 5,705 clusters. Only 1,749 clusters contains more than a single tuple, and only 1,488 contains at

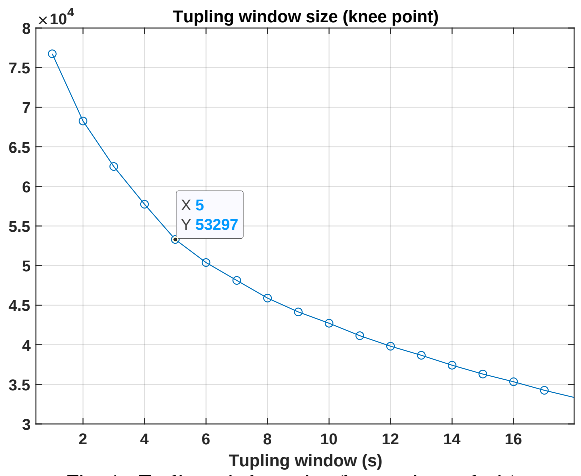

Fig. 4. Tupling windows size (knee point analysis).

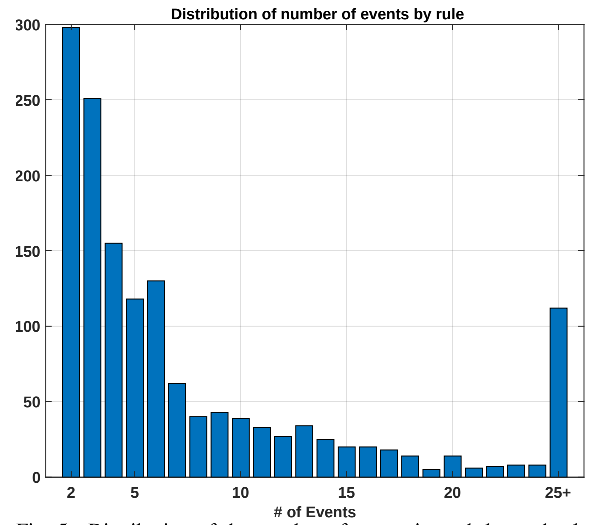

Fig. 5. Distribution of the number of events in each learned rule.

least two diagnostic events. Therefore, the proposal considers the 1,488 remaining clusters as potential filtering rules learned in an automatic way, with a reduction of about 97% with respect to the number of tuples. This allows understanding the advantage of using our proposal, since the analysts can focus directly on the review of the obtained rules (whose amount will be further reduced considering the scoring mechanism, as shown later) instead of digging in the large amount of tuples looking for relevant event correlations. Finally, Fig. 5 shows the distribution of the number of events for each learned rule. The 70% of the total rules learned is composed by less than 8 events. Rules with more than 20 events represent only less than 11%.

B. Filtering rule representation

As indicted in Section III-A, the resulting clusters are then converted in the Open-PSA fault-tree XML representation. As an example, Fig. 6 shows the tree representation of the rule obtained from the cluster capturing correlation indicated in the motivating example in Fig. 1, which should be analyzed by a domain expert. To facilitate the analysis of the expert, each node of the tree is enriched with the following information:

- an *eventID*, which is a combination of event name and event code extracted from the train model of the system, e.g., `81-_BREMG_A48`;
- the *time window* (for root event only), i.e., @window parameter, indicating the maximum time in which the rule applies;
- the *component* to which this event belongs to, i.e., @component parameter;
- the *probability* of occurrence, i.e., @probability parameter, which is defined as the ratio of number of times the event occurred in the cluster and the total number of tuples of the cluster;
- a natural language *description* of the event.

In the provided example, the cluster includes the events `81`, `383` and `410`, which have been grouped together by the proposal. These events happened at least two times within a time window of 1.6 seconds. The event type `BREMG_A48 (81)` has been indicated by the domain expert as a potential interesting event (through the dedicated list provided to the

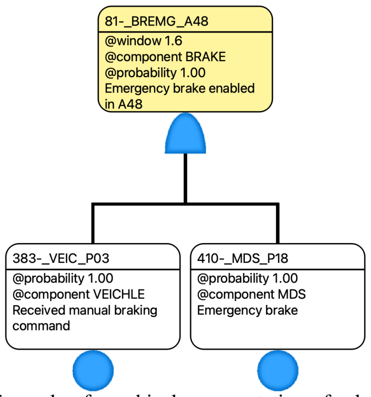

Fig. 6. Example of graphical representation of a learned rule.

FT building step). Therefore, it has been placed at the root of the fault-tree. The obtained tree represents an example of rule obtained in an automatic way by using the proposal, and that can be analyzed and modified by experts through any fault-tree editor compliant with the Open-PSA fault-tree XML representation. In this respect, the proposal allows to obtain potential filtering rules from historical data in an automatic way as well as to combine the view provided by logs with the expertise of the analysts.

C. Role of the score

The proposed score is adopted to (i) filter clusters that are not sufficiently supported by empirical evidence and to give (ii) a priority to each cluster. For instance, the `loss of communication with HVAC` event shown in the motivating example in Fig. 2 is high frequent in our dataset, making the probability to occur with another independent event sequence not negligible, i.e., a "collision" takes place. In this example, our approach detected the inconsistency within the example of Fig. 2, providing two filtering rules to the experts: the first including only the events with code `710` and `810`, with a score equal to 0.97; the second encompassing also the Code `14` event, but with a score equal to 0.03. Therefore, the very low score of the second rules suggests that the rule can be potentially discarded (as also confirmed by our industrial partner). Moreover, the higher score of the first rule provide to that rule an higher priority with respect to the second rule, as expected. The proposal detected also the correlation pointed out in Fig. 1, generating one candidate filtering rule (i.e., the one depicted in Fig. 6) with a 0.9 score, which suggests a strong correlation between events composing the rule. As already mentioned in Section III-B, it is also frequent that a cluster adds no new event combinations with respect to its child clusters. Hence, once assigned a score to the children, no score can be associated with the parent cluster. We handle this special case keeping only the parent cluster and discarding the children. This phenomena could be caused from either the truncation of the timing window or as effect of randomly event occurrence. In both cases, we delegate the validation to the expert domains keeping the parent cluster.

In TABLE I, we show how our score helps to reduce the number of clusters. For example, it can be noted that solely filtering the clusters from truncation/collision phenomena, we reduce the number of clusters of 40% (first row of the table).

TABLE I
ROLE OF THE SCORE

Score Threshold	Reduction (% clusters)
Filtering	40%
$Score > 0.2$	60%
$Score > 0.4$	75%
$Score > 0.6$	85%
$Score > 0.8$	90%

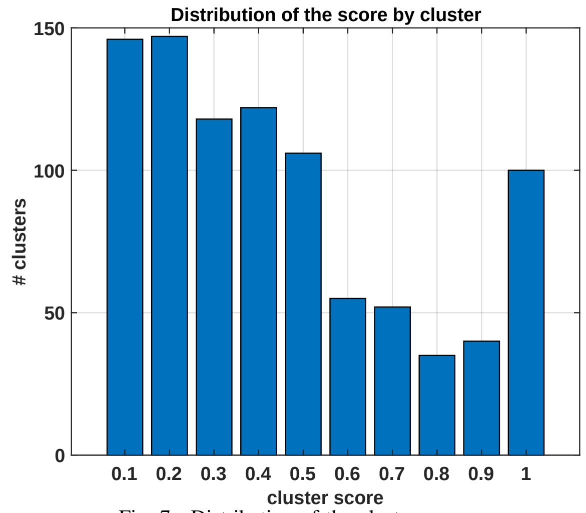

Fig. 7. Distribution of the clusters score.

In addition, we observed (with the support of our railway partner) that clusters with a low positive score do not support the filtering rules. For this reason, the domain expert can tune a threshold for the score; once set the threshold, the clusters with a score lower than the threshold are not presented as filtering rules candidates to the domain expert. In particular, the remaining rows of TABLE I provide the reduction factor with different score thresholds, which ranges from 60% to 90% for 0.2 and 0.8 threshold, respectively. This is also confirmed by the distribution of clusters score in Fig. 7, which shows that most of the clusters have a score lower than 0.5. Experts analyzed 20% of the clusters with a score of 0.4 and concluded it could be used as a valuable threshold. In conclusion, we started from more than six thousand unique tuples. Through the clustering, we reduce the number of unique tuples to 75%, and, subsequently, choosing a threshold of 0.4, we reduce the clusters of 75% with 388 remaining clusters, getting a 93% final reduction of the unique tuples.

V. Related Work

Diagnostic logs contain a large amount of textual and numerical information about system behavior under real workload conditions, which makes their analysis useful for classifying the propagation of errors and failure modes. Leveraging **coalescence techniques**, like heuristic tupling, the error events can be coalesced in tuples to associate them with a failure mode. For instance, error events occurring close in time can be coalesced to represent a single failure mode. The validity of heuristic models for time coalescence in event logs is discussed in [13], which also presents a sensitivity analysis that is adopted in our study. Spatial coalescence heuristics are adopted in the analysis of large systems such as data-centers and supercomputers [14], [15]. Collisions are the main issue of tupling heuristics: errors of different failure modes can be associated to the same failure mode, for example, because the chance that independent failures occur on different nodes is not negligible or due to a wrong tuning of the time window. In this paper, the tupling heuristics are adopted to identify independent events root causes, while we adopted clustering to support the filtering rules represented by the tuples.

Clustering techniques group events that are similar each other and dissimilar between the events of other clusters. Clustering methods includes Partitional and Hierarchical clustering [16]. *Partitional* clustering defines at-priori the number of clusters, and searches the partition maximizing a given function cost. An example is k-means [17], which tries to minimize the total intra-cluster variance at each iteration. *Hierarchical clustering* can be divisive or agglomerative. Initially, agglomerative algorithms assume each cluster (leaf) containing a single object; subsequently, at each iteration, the "closest" clusters are joined to get a larger cluster. Measures of similarity between clusters are necessary to link the clusters. Linkage methods, such as Single-link (used in our proposal), Average-link and Complete-link, calculate the inter-cluster distance considering all combinations of points between the two clusters [18], [19], while geometric methods adopt geometric centers to represent the clusters, and to calculate the inter-cluster distance. For instance, the Ward's method is a geometric method that minimizes the intra-cluster variance. Divisive algorithms instead initially consider a partition formed by a single large cluster containing all the elements; then, the cluster is divided iteratively. Divisive clustering, as opposed to agglomerative one, needs to measure the density or sparsity of points within a cluster to decide if proceed with the division.

Discovering filtering rules allows obtaining valuable information about events correlation with the aim to pinpoint interesting events from data under analysis. Different approaches have been proposed with aim to infer event correlations directly from data, making assumptions on the data set, conducting statistical tests and representing correlation as fault-tree models [8]–[10], [20], [21]. In particular, the work in [8] is the first completely automated tool to test the causality in the fault-tree construction statistically. However, differently from our proposal, the existing approaches require labeled datasets, i.e., dataset where events are already grouped together and the label indicated the occurrence or not of the root event is provided, which can be difficult to obtain for real-world production system, like our railway case study. Similar to our approach, the work in [22] tries to infer filtering rules (for outlier detection) from unlabeled data; however, the approach combines both unsupervised and supervised approaches to discover filtering rules, hence requires a labeling step before inferring the rules.

VI. Conclusion

The paper proposed an unsupervised approach to discover filtering rules from diagnostic logs. The approach allows to infers potential events correlations in an automatic way, extracting filtering rules represented as fault-trees. Filtering rules are accompanied with a novel score allowing to support the validation of rules. The approach has been applied in a preliminary railway case study, with more than 710k events generated by on-board train equipment during eight weeks of operation. Future work will focus on extending the proposal to automatically suggest primary events for discovered rules as well as on the extensive validation of the approach in different industrial contexts.

Acknowledgment

This work has been supported by the Meditech Competence Center under the AID4TRAIN Project (I65F21001010005).

References

[1] M. Cinque, R. Della Corte, and A. Pecchia, "An empirical analysis of error propagation in critical software systems," *Empirical Software Engineering*, vol. 25, no. 4, pp. 2450–2484, 2020.

[2] J. Tian, S. Rudraraju, and Z. Li, "Evaluating web software reliability based on workload and failure data extracted from server logs," *Soft Eng, IEEE Transactions on*, vol. 30, no. 11, pp. 754–769, Nov 2004.

[3] E. Chuah, A. Jhumka, J. C. Browne, B. Barth, and S. Narasimhamurthy, "Insights into the diagnosis of system failures from cluster message logs," in *11th European Dependable Computing Conference (EDCC)*, Sept 2015, pp. 225–232.

[4] A. Pecchia, I. Weber, M. Cinque, and Y. Ma, "Discovering process models for the analysis of application failures under uncertainty of event logs," *Knowledge-Based Systems*, vol. 189, p. 105054, 2020.

[5] H. Lal and G. Pahwa, "Root cause analysis of software bugs using machine learning techniques," in *2017 7th International Conference on Cloud Computing, Data Science & Engineering-Confluence*. IEEE, 2017, pp. 105–111.

[6] J. Goikoetxea, "Shift2rail connecta: The next generation of the train control and monitoring system." Zenodo, Apr 2018.

[7] M. Cinque, R. Della Corte, and A. Pecchia, "Contextual filtering and prioritization of computer application logs for security situational awareness," *Future Generation Computer Systems*, vol. 111, pp. 668–680, 2020.

[8] M. Nauta, D. Bucur, and M. Stoelinga, "Lift: Learning fault trees from observational data," in *QEST*, 2018.

[9] A. Linard *et al.*, "Fault trees from data: Efficient learning with an evolutionary algorithm," *ArXiv*, vol. abs/1909.06258, 2019.

[10] A. Linard, M. L. P. Bueno, D. Bucur, and M. Stoelinga, "Induction of fault trees through bayesian networks," *Proceedings of the 29th European Safety and Reliability Conference (ESREL)*, 2019.

[11] C. Di Martino, M. Cinque, and D. Cotroneo, "Assessing time coalescence techniques for the analysis of supercomputer logs," in *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)*. IEEE, 2012, pp. 1–12.

[12] D. Xu and Y. Tian, "A comprehensive survey of clustering algorithms," *Annals of Data Science*, vol. 2, no. 2, pp. 165–193, 2015.

[13] J. P. Hansen and D. P. Siewiorek, "Models for time coalescence in event logs," *Digest of Papers. FTCS-22: The Twenty-Second International Symposium on Fault-Tolerant Computing*, pp. 221–227, 1992.

[14] A. J. Oliner and J. Stearley, "What supercomputers say: A study of five system logs," *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, pp. 575–584, 2007.

[15] Y. Liang, Y. Zhang, A. Sivasubramaniam, M. A. Jette, and R. K. Sahoo, "Bluegene/l failure analysis and prediction models," *International Conference on Dependable Systems and Networks*, pp. 425–434, 2006.

[16] R. Xu and D. C. Wunsch, "Survey of clustering algorithms," *IEEE Transactions on Neural Networks*, vol. 16, pp. 645–678, 2005.

[17] J. MacQueen, "Some methods for classification and analysis of multivariate observations," 1967.

[18] P. H. A. Sneath, "The application of computers to taxonomy." *Journal of general microbiology*, vol. 17 1, pp. 201–26, 1957.

[19] T. Sørensen *et al.*, "A method of establishing group of equal amplitude in plant sociobiology based on similarity of species content and its application to analyses of the vegetation on danish commons," 1948.

[20] P. J. Nolan, M. G. Madden, and P. R. Muldoon, "Diagnosis using fault trees induced from simulated incipient fault case data," 1994.

[21] P. D. McNicholas, T. B. Murphy, and M. O'Regan, "Standardising the lift of an association rule," *Comput. Stat. Data Anal.*, vol. 52, pp. 4712–4721, 2008.

[22] K. Yamanishi and J.-i. Takeuchi, "Discovering outlier filtering rules from unlabeled data: Combining a supervised learner with an unsupervised learner," in *Proc. of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '01. New York, NY, USA: Association for Computing Machinery, 2001, p. 389–394.