WILEY

ALACA: A platform for dynamic alarm collection and alert notification in network management systems

Selçuk Emre Solmaz[1] 🅾 | Buğra Gedik[1] | Hakan Ferhatosmanoğlu[1] | Selçuk Sözüer[3] |
Engin Zeydan[2] | Çağrı Özgenç Etemoğlu[2]

[1]Bilkent University, Ankara, Turkey

[2]Türk Telekom Labs, Istanbul, Turkey

[3]OBSS, Istanbul, Turkey

Correspondence
Selcuk Emre Solmaz, Bilkent University,
Ankara, Turkey.
Email: selcuksolmaz@bilkent.edu.tr

Funding information
TUBITAK TEYDEB 1501, Grant/Award
Number: 3130411

Summary

Mobile network operators run Operations Support Systems that produce vast amounts of alarm events. These events can have different significance levels and domains and also can trigger other ones. Network operators face the challenge to identify the significance and root causes of these system problems in real time and to keep the number of remedial actions at an optimal level, so that customer satisfaction rates can be guaranteed at a reasonable cost. In this paper, we propose a scalable streaming alarm management system, referred to as Alarm Collector and Analyzer, that includes complex event processing and root cause analysis. We describe a rule mining and root cause analysis solution for alarm event correlation and analyses. The solution includes a dynamic index for matching active alarms, an algorithm for generating candidate alarm rules, a sliding window–based approach to save system resources, and a graph-based solution to identify root causes. Alarm Collector and Analyzer is used in the network operation center of a major mobile telecom provider. It helps operators to enhance the design of their alarm management systems by allowing continuous analysis of data and event streams and predict network behavior with respect to potential failures by using the results of root cause analysis. We present experimental results that provide insights on performance of real-time alarm data analytics systems.

KEYWORDS

Alarm, Alarm Rule Mining, Alarm Management, Data Analytics, Root-Cause

1 | INTRODUCTION

In a large-scale network operations center, millions of alarm events are generated typically each day. Many of the alarms in the system may have no significance and need to be filtered out at the source level. The important ones need to be identified and resolved immediately. All nodes in the system and their status update information need to be gathered and analyzed in real time on a regular basis. Therefore, scalable and intelligent alarm management is becoming a necessity for any network operation and data center.[1,2] An ideal alarm informa-

tion management system needs to identify and evaluate what is occurring in the system, to find plausible root causes for alarms and to take appropriate actions effectively and efficiently. As the underlying systems are growing both in size and complexity, the alarm management itself is becoming a central bottleneck to enable a seamless operation.

Traditionally, network operators have back-office teams to determine significance of alarms using their domain expertise. However, the telecommunication field has many subdomains, such as transmission (microwave, Synchronous Digital Hierarchy, etc), radio access (Base Transceiver Station,

nodeB. eNodeB), and core (Mobile Switching Center, IP Multimedia Subsystem, Gateway GPRS Support Node, Serving GPRS Support Node, etc.) networks. Moreover, additional management, service, and security layers bring extra elements on top of these domains. Gaining experience on each subdomain requires many work hours over years. Moreover, the total reliance on human operators for discovering significance of alarms, as well as the correlations between them, has proved to be difficult and inconsistent. Hence, there is a clear need for developing autonomous analytic solutions to support operators in providing quick response and fusion of accumulated alarm information. Most of the time, network faults are generally handled or cleared using trouble ticket management systems (eg, HP TEMIP[3]), which works together, typically, with an asset management system (eg, IBM Maximo[4]). However, lack of intelligent network alarm management systems exploiting rules derived through data mining causes problems in performance of traditional trouble ticket management systems of mobile operators, thus resulting in unsolicited site visits.

In this paper, we propose a platform solution to analyze millions of alarms in real time, produce alerts, and analyze the root causes for a large-scale network operation data center. Our system includes methods for complex event processing (CEP) and root cause analysis, where millions of alarms are captured, correlated, and mined for root causes in real time. We have developed our system inside a major mobile operator with 16 million subscribers to analyze the root causes of alarms in the network operation center. As a major mobile telecom provider, it is running a nationwide network of Global System for Mobile Communications/Universal Mobile Telecommunications System/Long Term Evolution (LTE) infrastructure, consisting of different types of nodes. The most common system types are the following.

1. Base Station Subsystem: the base stations and their controllers
2. Network and Switching Subsystem: the part of the network most similar to a fixed network, sometimes just called the "core network"
3. Packet Core Network: the part which allows packet-based Internet connections
4. Operations Support System (OSS): network maintenance

In the presence of alarms arriving from multiple heterogeneous sources, it is a major challenge to identify the root causes of problems in real time that can keep the number of actions (in terms of ticket generation, site visits, etc) low. To adapt to changes in the network system, the methodologies for building new rules need to be revised in an adaptive manner. The proposed platform targets at processing network alarm information in real time and includes novel data analytics approaches to add more capability to traditional alarm management systems. Thus, mobile operators can make

highly accurate and on-time decisions in the presence of overwhelming number of alarms. Benefits of this work to mobile operators and data centers include a new ranking of alarm significance, recognizing inherent correlation between alarms, and root cause analysis for fast and accurate decisions resulting in less site visits and ticket generations. Our solution can also be used for trend analysis of alarms on a seasonal scale to realize efficient, highly available network infrastructure investments and enhance operational processes resulting in long-term cost savings.

In summary, the main contributions of this paper are as follows:

1. A streaming big data infrastructure is used for high availability and scalability, while managing and analyzing vast amount of alarm data.
2. A rule mining algorithm with probabilistic candidate rule filtering is proposed to create alarm rules, saving system resources.
3. A root cause algorithm is proposed to find influential alarms in the system.

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 defines our methodology. Section 4 presents our infrastructure and solution. Section 5 presents our experience with the rule mining algorithm. Section 6 contains concluding remarks.

2 | RELATED WORK

Alarm correlation for alarm flow analysis inside telecommunication networks is discussed in a few earlier works[5-11] (see also Meira[12] for an overview). There are also works that study temporal data mining (see Laxman and Sastry[13] for a survey). Some of these papers consider sequential data mining techniques that find the most frequent event sequences in a sequence database.[8,14] GSP[14] and WINEPI[8] are among the first algorithms to apply Apriori algorithm[15] to find sequential association rules temporally. These algorithms usually require a user-defined sliding-time window duration to traverse the data. Most of the sequential data mining methods proposed so far are based on discovering temporal relations between events in discrete time series.[14] These methods are based on finding the frequencies of sequences of events and generating the rule candidates against the database.[16] Others concentrate on prediction problems for sequential rule mining on several different application domains. Some of these approaches are based on rule correlation, fuzzy logic, coding correlation, Bayesian networks, and artificial neural networks.[8,16-19]

Each of the aforementioned approaches, ranging from generating an alarm correlation engine to alarm modeling and validation, have their own advantages and disadvantages.

Alarm modeling approaches that provide explanations for the monitored resources and the associated alarms are studied in Frohlich et al and Yemini et al.[20,21] Mannila et al propose an algorithm for finding sequential rules in a single sequence of alarm events using frequent episode discovery.[8] Klemettinen et al use a semiautomatic approach to find a complementary alarm management solution.[9] In Wietgrefe et al,[10] the authors consider finding the root cause alarm using neural networks. Wallin et al aim to establish a general state machine for the alarms and propose various interpretations of the alarm state based on the context information in Wallin[6] and focus on validating the alarm model based on the static analysis in Wallin.[22] Duarte et al propose an SNMP focused specific language called ANEMONA that focuses on alarm monitoring solutions.[7] Li et al propose a neural network and FP-tree approach to perform weighted association rule mining in the alarm domain.[11] The authors in Çelebi et al[23] propose a new alarm correlation, rule discovery, and significant rule selection method based on applying sequential rule mining algorithm for alarm sequences based on analysis of real data collected from a mobile telecom operator with an additional parameter called *time-confidence*. The paper in Sozuer et al[24] investigates a new clustering approach by applying 2 new metrics borrowed from document classification approaches to increase the accuracy of the sequential alarm rules. In our recent demo paper,[25] we have demonstrated a real-time data analytics platform where alarm/event data analyses are performed by capturing, processing, and visualizing alarms via appropriate notifications in real time.

Klaus Julisch proposed to cluster intrusion detection alarms to support root cause analysis in Julisch.[26] They develop a novel alarm-clustering method, which groups similar alarms together, assuming these alarms also share the same root cause called *generalized alarm*. They focused on alarm-clustering problem more than defining root cause. Zong et al study root cause analysis within the context of alarms.[27] They propose algorithms that use effective pruning and sampling methods for fast critical alert mining. They mine k critical alerts from alert graphs on demand. Man et al use a big data infrastructure to analyze network alarms.[28] They use FP-growth algorithm for alarm correlation analysis and data mining. They show that correlation mining for 8000 network elements on a 143 node PC server cluster takes 20 minutes. In contrast to FP-growth algorithm, we propose an algorithm to identify root causes using less memory. The authors in Bellec[29] study the problem of analyzing, interpreting, and reducing the number of these alarms before trying to localize the faults leveraging clustering approaches. The authors in Makanju et al[30] propose a hybrid framework by allowing High Performance Clusters for alert detection in system logs.

Compared to the mentioned works, our main contribution is to create alarm rules to identify root causes by developing a system that combines a streaming big data infrastructure, a dynamic index for matching active alarms, an algorithm for generating candidate alarms, and a sliding window–based approach to save system resources. Table 1 gives main alarm correlation models for the considered problem, highlighting some of the most relevant works for each problem, while situating the contributions of this work.

3 | METHODOLOGY

A large mobile telecommunications provider needs an intelligent alarm management platform. Such a platform would collect event data related to alarms and perform alarm analysis on this data to mine alarm rules and determine influential alarms. In this paper, we present the Alarm Collector and Analyzer (ALACA) platform, which is designed to handle

TABLE 1 A comparison of various alarm correlation techniques applied for mobile operator networks

	Traditional approaches		Proposed approach	
	Characteristics	Limitations	Advantages	Differences
Sequential data mining	WINEPI[8] for discovery	User-defined sliding-time	Cost-effective Real-time	Root cause analysis
			Uses less memory	Rule formation
	GSP[14] using Apriori FP-growth and and big data[28]	Window duration High memory for large number of nodes		
Root cause analysis	Critical alert mining[27]	Finds fixed k critical alarms Focuses on pruning and sampling	Not fixed k root causes	Alert notification rather than on-demand critical rule mining
Cluster-based techniques	TF.IIF clustered[24] High performance clusters[30] Generalized alarms[26]	Focuses on clustering rather than defining fault diagnosis	Uses all data	Rule formation Nonclustered
Proprietary vendor-specific solutions	IBM Maximo[4] HP TEMIP[3]	Vendor lock-in New service initiation New feature support	Cost-effective Open-source Scalable	Rule formation Root cause analysis

these tasks. The ALACA platform relies on a big data architecture to store alarm event data, a stream processing module for complex event analysis, a root cause analysis module to mine alarm rules and detect influential alarms, and finally a Web client for end-user interaction. We present the ALACA platform's architecture later in Section 4.

In this section, we formally describe the alarms and rules used in this work. We formalize the semantics of our rules by formulating the matching between a rule and an alarm sequence. We give the support and confidence requirements of our rules. Finally, we describe our rule mining algorithm, including a probabilistic filtering technique that significantly reduces the algorithm's memory requirement and improves its running time performance.

We use the following conventions throughout the paper with respect to notation: Symbols in regular font represent scalars, whereas those in boldface font represent vectors. Sets are denoted by uppercase calligraphic symbols.

3.1 | Formalization

Each alarm has a kind and a time interval during which it was active. For example, an alarm can be specified as $A = (X, [t_0, t_1])$. Here, X is the alarm kind and $[t_0, t_1]$ is the time interval during which the alarm was active. The start time sorted list of all alarms is denoted as $\mathcal{D}$. The alarm kind for an alarm A is denoted as $A.k$ and the set of all alarm kinds as $\mathcal{K} = \{A.k : A \in \mathcal{D}\}$. The alarm interval is denoted as $A.i$. The start time of the alarm interval is denoted as $A.t_s$ and the end time as $A.t_e$. Sorted alarms of kind k are denoted as $\mathcal{L}$.

Alarm rules are defined as logical implications, where the antecedent is an ordered set of alarm kinds and the consequent is a single alarm kind. Each alarm rule is also associated with a time duration. For instance, a rule can be specified as $R = [X, Y] \rightarrow Z : t$. Here, there are 2 alarm kinds, namely, X and Y, appearing as an ordered set on the antecedent, and there is a single alarm kind Z appearing as the consequent. Time duration, t, is given as the rule duration.

The meaning of an alarm rule is as follows: When the alarms specified on the antecedent occur in order, then the alarm specified on the consequent is expected to occur within t time units. As an additional requirement, the intervals of all the alarms involved in a rule should intersect. For instance, in the running example, first, X should start. Then Y should start before X ends. Then the rule predicts that, after Y starts, Z is expected to start within t time units, before X and Y end. For a rule R, we denote the antecedent as $R.a$ and the consequent as $R.c$. The duration of the rule is denoted by $R.d$. The consequent cannot be contained in the antecedent, that is $R.c \notin R.a$. Figure 1 illustrates a sequence of events matching a rule.
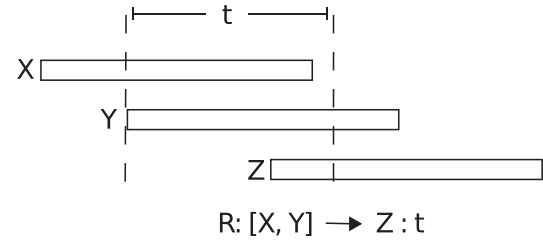

FIGURE 1 An example sequence of events that match a given rule

Formally, a set of events ordered by their start time, denoted as $\mathcal{U}$, match a rule R, based on the following definition of a Boolean match function:

$$
\begin{aligned}
match(R, \mathcal{U}) \equiv & \bigcap_{A \in \mathcal{U}} A.i \neq \emptyset \ \wedge \\
& [A.k : A \in \mathcal{U}] = R.a \cup [R.c] \ \wedge \\
& \mathcal{U}_{[-1]}.t_s - \mathcal{U}_{[-2]}.t_s \leqslant R.d
\end{aligned} \tag{1}
$$

In Equation 1, there are 3 conditions: First condition requires that the intersection of the alarm intervals in $\mathcal{U}$ cannot be an empty set. Second condition requires that the set of all alarm kinds in $\mathcal{U}$ should be equal to the union of the alarm kind set of the rule's antecedent alarm kinds set and the consequent alarm kind. We use $_{[j]}$ notation to index the time ordered event set, where an index $j = -1$ represents the last item and -2 the penultimate one. Final condition requires that the difference between the last item's start time in $\mathcal{U}$ and the penultimate item's start time should be less than or equal to the rule R's duration. Therefore, for a rule match, these 3 conditions of Equation 1 should be true.

3.2 | Mining alarm rules

To mine alarm rules from a data set of alarms, we develop a scalable rule mining algorithm. The algorithm can mine rules in the format outlined so far, which additionally satisfy given confidence and support requirements. The algorithm can also be trivially adapted to mine rules where the antecedent is an unordered set. We denote the support threshold as $\tau_s \in \mathbb{N}^+$ and the confidence threshold as $t_c \in (0, 1]$. To satisfy the support requirement, there must be at least τ_s distinct event subsequences in the data set that match the rule. To support the confidence requirement, at least t_c fraction of the event sequences that match the antecedent part of the rule must be followed by an addition event that results in the extended sequence matching the entire rule.

We formally define the support of a rule as follows, recalling that $\mathcal{D}$ is the time ordered set of all events:

$$
support(R, \mathcal{D}) = |\{\mathcal{U} \subseteq \mathcal{D} : match(R, \mathcal{U})\}|. \tag{2}
$$

We formally define the confidence of a rule as follows:

$$confidence(R, D) = support(R, D) \ /$$
$$|\{\mathcal{U} \subseteq \mathcal{D} : \underset{A \in \mathcal{U}}{\cap} A.i \neq \emptyset \wedge [A.k : A \in \mathcal{U}] = R.a\}|. \tag{3}$$

Before we apply the rule mining algorithm, the alarm data set is taken through a preprocessing, which we detail next.

first partition alarms based on their kinds. Then for each alarm kind, we find a set of candidate rules whose consequent is the given alarm kind (line 4). This is done via the MINEALARMRULESFORKIND procedure. The set of candidate rules found this way satisfy the support requirement but may not satisfy the confidence requirement. Thus, the candidate rules go through a filtering process (line 5). This is done via the ENFORCECONFIDENCES algorithm.

Algorithm 1: Alarm pre-processing.

Input: $\mathcal{D}$, Sorted alarms; $\mathcal{K}$, Alarm kinds
Output: $\mathcal{C}$, Pre-processed alarms; $\mathcal{I}$, Interval index

1   $M \leftarrow \{u \mapsto \mathcal{L} : u \in \mathcal{K} \wedge \mathcal{L} = \{A \in \mathcal{D} : A.k = u\}\}$
2   **for** $u \mapsto \mathcal{L} \in M$ **do**
3     $n \leftarrow 0$
4   **for** $i \in [1..|\mathcal{L}|)$ **do**   ▷ in start time order
5   **if** $\mathcal{L}_{[i]}.t_s < \mathcal{L}_{[n]}.t_e$ **then**
6   **if** $\mathcal{L}_{[i]}.t_e > \mathcal{L}_{[n]}.t_e$ **then**
7           $\mathcal{L}_{[n]}.i = [\mathcal{L}_{[n]}.t_s, \mathcal{L}_{[i]}.t_e]$
8   **else**
9         $n \leftarrow n + 1$
10        $\mathcal{L}_{[n]} \leftarrow \mathcal{L}_{[i]}$
11    $\mathcal{L} \leftarrow \mathcal{L}_{[0..n]}$
12   $\mathcal{C} \leftarrow sort(\cup_{u \mapsto \mathcal{L} \in M} \mathcal{L}, \_ \mapsto \_.t_s)$
13   $\mathcal{I} \leftarrow build\_interval\_index(\mathcal{C})$

3.2.1 │ Alarm preprocessing

The preprocessing phase is used to cleanse the alarm data set and to index the alarms. The cleansing involves merging alarms of same kind whose time intervals intersect and ordering the resulting alarm data set based on start time. Indexing involves creating an *interval tree* using the alarm intervals. The interval tree[31] enables querying the alarm data set using a query time point. In particular, it can retrieve the alarms whose time intervals contain the query time point in $\mathcal{O}(\log(n))$ time, where *n* is the number of alarms. An interval index can be constructed in $\mathcal{O}(n \cdot \log(n))$ time. Algorithm 1 describes the preprocessing step.

The filtering process removes the candidate rules that do not meet the confidence criteria. Merging rules generated for different alarm kinds produce the final set of rules.

MINEALARMRULESFORKIND procedure, given in Algorithm 3, describes how we detect alarm rules for a given alarm kind. We first do a quick check to see if the number of alarms of the given kind is at least equal to the support threshold. If not, we cannot have enough occurrences for a rule whose consequent is of the given kind. Thus, we simply return in this case. Otherwise, to determine the candidate rules, we iterate over the alarms of the given kind. This time ordered alarm list is already computed earlier and is the

---

Algorithm 2: MINEALARMRULES algorithm

Input: $\mathcal{C}$, Pre-processed alarms; $\mathcal{I}$, Interval index; τ_s, Support threshold; τ_c, Confidence threshold
Output: $\mathcal{R}$, Mined alarm rules

1   $\mathcal{R} \leftarrow \varnothing$
2   $M \leftarrow \{u \mapsto \mathcal{L} : u \in \mathcal{K} \wedge \mathcal{L} = \{A \in \mathcal{C} : A.k = u\}\}$
3   **for** $u \mapsto L \in M$ **do**
4     $\mathcal{V} \leftarrow$ MINEALARMRULESFORKIND$(\mathcal{L}, \mathcal{I}, \tau_s)$
5     $\mathcal{R} \leftarrow \mathcal{R} \cup$ ENFORCECONFIDENCES$(\mathcal{V}, \mathcal{C}, \mathcal{I}, \tau_c)$

3.2.2 │ Rule detection

MINEALARMRULESFORKIND procedure, given in Algorithm 2, describes how we detect alarm rules. We

parameter $\mathcal{L}$ to the algorithm in the pseudocode. For each alarm in this list, referred to as an *anchor alarm*, we generate a number of candidate rules (line 5). These candidate rules have the important property that for each one, there

---

Algorithm 3: MINEALARMRULESFORKIND algorithm

Input: $\mathcal{L}$, Sorted alarms of kind u; $\mathcal{I}$, Interval index; τ_s, Support threshold
Output: $\mathcal{R}$, Candidate alarm rules

1   $\mathcal{R} \leftarrow \varnothing$
2   **if** $|\mathcal{L}| < \tau_s$ **then return**
3   $M \leftarrow \varnothing$ ▷ map of candidate rules
4   **for** $A \in \mathcal{L}$ **do**
5   **for** $R \in$ GETCANDIDATERULES$(A, \mathcal{I})$ **do**
6   **if** $R.a \in M$ **then**
7   Let $M[R.a] = (R', s)$
8         $R.d \leftarrow max(R.d, R'.d)$
9         $M[R.a] \leftarrow (R, s+1)$
10   **else**
11         $M[R.a] \leftarrow (R, 1)$
12   **for** $K \mapsto (R, s) \in M$ **do**
13   **if** $s \geq \tau_s$ **then**
14       $\mathcal{R} \leftarrow \mathcal{R} \cup R$

---

is a matching alarm subsequence terminating at the anchor alarm. The candidate rules are found via the GETCANDIDATERULES procedure. As we iterate through the anchor alarms and generate candidate rules, we use a map, M in the pseudocode, to keep a count of the number of times each candidate rule has appeared. Two candidate rules returned by different calls to GETCANDIDATERULES are considered the same if their antecedents are the same (their consequent are the same by construction). They may differ in their duration. We take the maximum duration as the duration of the candidate rule kept in the map for counting the occurrences. Alternatively, a statistical choice could be made, such as maintaining a running mean and average duration and at the end determining the duration value that corresponds to the 95% percentile assuming a normal distribution. At the end of the process, we only keep the candidate rules whose occurrence counts are greater than or equal to the support threshold.

Algorithm 4: GETCANDIDATERULES algorithm

Input: A, Alarm; $\mathcal{I}$, Interval index
Output: $\mathcal{R}$, Candidate alarm rules

1   $\mathcal{R} \leftarrow \varnothing$
2   $\mathcal{V} \leftarrow \varnothing$
3   **for** $A' \in \mathcal{I}.lookup(A.t_s)$ **do**
4   **if** $A'.t_s \neq A.t_s$ **and** $A'.t_e \neq A.t_s$ **then**
5       $\mathcal{V} \leftarrow \mathcal{V} \cup \{A'\}$
    ▷ For time ordered subsets of the antecedents
6   **for** $\mathcal{U} \subset \mathcal{V}$ **do**
7   **if** $|\mathcal{U}| = 0$ **then continue**
8     $t \leftarrow A.t_s - \mathcal{U}_{[-1]}.t_s$
9     $R \leftarrow [A'.k : A' \in \mathcal{U}] \rightarrow A.c : t$
10    $\mathcal{R} \leftarrow \mathcal{R} \cup \{R\}$

GETCANDIDATERULES procedure, given in Algorithm 4, describes how we generate the list of candidate rules for a

particular anchor event. These candidate rules are the ones for which there exists a matching alarm subsequence terminating at the anchor event. For this purpose, we query the interval tree with the start time of the anchor event and find all other events that are active at that time. The result is a subset of events ordered based on their start time. Any start time ordered subset of these rules can serve as the antecedent of a candidate rule. Thus, we generate one candidate rule for each such subset, with the duration set to the time difference between the start times of the last event in the subset and the anchor event.

Finally, ENFORCECONIDENCES procedure, given in Algorithm 5, describes how we filter the short list of candidate alarms to retain only the ones that satisfy the confidence requirement. For this purpose, we compute the confidence of each rule. The last element of the rule's antecedent is used to iterate over all the events of that kind. Each one of these events represent a possible point of match for the rule's antecedent. Let us call these events *vantage events*. We use a vantage event's start time to query the interval index and find all other events that were active during that time. We iterate over the results from querying the interval tree in start time order and check if we can find a sequence of events whose kinds match the rule's antecedent up to and excluding its last item. If so, then we have an antecedent match at this vantage event and we increment the corresponding counter, n_s in the pseudocode. An antecedent match may correspond to a rule match or not. To check that, we simply find out if there is an event whose kind matches that of the rule consequent. We limit our search to events that start after the leverage event within the rule's duration. Also, it should start before any of the events matching the antecedent finish. If we find a rule match, then we increment the corresponding counter, n_t in the pseudocode. Accordingly, n_t/n_s gives us the confidence, which is compared against the confidence threshold, τ_c, to perform the filtering.

Algorithm 5: ENFORCECONFIDENCES algorithm

Input: $\mathcal{V}$, Candidate rules; $\mathcal{C}$, Pre-processed alarms; $\mathcal{I}$, Interval index; τ_c, Confidence threshold

Output: $\mathcal{R}$, Alarm rules

1 $\mathcal{R} \leftarrow \varnothing$

2 **for** $R \in \mathcal{V}$ **do**

3      $n_s \leftarrow 0;\; n_t \leftarrow 0$

4   **for** $i \in [0..|\mathcal{C}|)$ **do**

5   **if** $\mathcal{C}_{[i]}.k \neq R.a_{[-1]}$ **then continue**

          ▷ check antecedent alarms

6         $j \leftarrow 0;\; t \leftarrow \mathcal{C}_{[i]}.t_e$

7   **for** $A \in \mathcal{I}.lookup(\mathcal{C}_{[i]}.t_s)$ **do**

8   **if** $R.a_{[j]} = A.k$ **then**

9               $j \leftarrow j + 1$

10               $t \leftarrow min(t, A.t_e)$

11   **if** $j \neq |R.a| - 1$ **then continue**

12         $n_s \leftarrow n_s + 1$

          ▷ check consequent alarm

13   **for** $l \in [i + 1..|\mathcal{C}|)$ **do**

14   **if** $\mathcal{C}_{[l]}.t_s > \mathcal{C}_{[i]}.t_s + R.d$ **or** $\mathcal{C}_{[l]}.t_s >= t$ **then**

15   **break**

16   **if** $\mathcal{C}_{[l]}.k = R.c$ **then**

17               $n_t \leftarrow n_t + 1$

18   **break**

19   **if** $n_t/n_s \geq \tau_c$ **then**

20         $\mathcal{R} \leftarrow \mathcal{R} \cup \{R\}$

The alarm rules identified as above are then transformed into a directed graph $\langle V, E \rangle$, where vertexes V are the alerts and the edges E are the rule relationships between them. For example, if 2 rules are calculated for a network equipment, such as $[A, B] \rightarrow C$ and $[D, E] \rightarrow B$, then the vertex set becomes $V = \{A, B, C, D, E\}$ and the edge set becomes $E = \{(A, C), (B, C), (D, B), (E, B)\}$. Graphs for unique network equipment types or the entire system can be generated. Based on the graph representation, root cause alarms are identified by a scoring mechanism. After calculating the scores of all alarms, most influential alarms are detected as the ones with the top influence scores.

each vertex, and thus each unique alarm kind that appears in the graph, accumulates some number of points. To compute the global influence of an alarm, we sum up all the points it has accumulated across all composite graphs.

3.3 | Probabilistic candidate rule filtering

A downside of our rule mining algorithm is that the memory usage can go up significantly, since we maintain many candidate rules in Algorithm 3 with very low occurrence counts that are eventually discarded because of failing the support requirement. This may prevent the algorithm scaling

---

Algorithm 6: SCORE algorithm calculates each alarm's influence score

Input: A directed graph, G(V,E)

1 **foreach** V_i **in** G **do**

2   **if** V_i *is a leaf node* **then**

3         $V_i.s \leftarrow 0$

4   **else**

5         $V_i.s \leftarrow 1 + \texttt{sum}\,(V_{ij}.s \mid V_{ij}$ *is an out neighbor of* $V_i)\,/2$

The alarm influence score calculation is performed as follows: First, for each network equipment (ie, Node Name in Table 2) that generates alarms, a composite graph is created by combining the rules mined for that equipment. Then this composite graph is topologically sorted. Next, vertexes of the graph are processed in reverse topological order and scored by Algorithm 6. When a composite graph is fully processed,

TABLE 2 Alarm *Event* data format

Field	Explanation
Node Name	Network equipment producing the alarm
Alarm Name	Name identifying the alarm kind
Event Time	Time duration of the alarm
Severity	Priority state of the alarm

to large alarm data sets. To solve this problem, we have developed a probabilistic filtering algorithm. The fundamental idea behind the filtering algorithm is that, if the current occurrence count of a rule is very low compared to the expected occurrence count to eventually pass the support threshold assuming a uniform distribution of occurrences over the alarm set, then that candidate rule can be discarded right away. We mathematically formalize this filtering concept.

Let S_n be a random variable representing the number of occurrences of a candidate rule after observing n anchor events. We assume that occurrence of a rule for the ith event is a Bernoulli random variable, denoted as X_i. To have enough support in the average case, we need to have $E[X_i] \geqslant \tau_s/N, \forall i$, where N is the total number of anchor events ($|\mathcal{L}|$ in the pseudocode). We have $S_n = \sum_{i=1}^{N} X_i$ and $E[S_n] = n \cdot \tau_s/N$. The probabilistic candidate rule filtering works by discarding a candidate rule if after observing n anchor events, S_n is less than k and $Pr(S_n \leqslant k) \leqslant 1 - \phi$, where $\phi \in [0, 1]$ is a configuration parameter called the *filtering confidence*. ϕ is typically close to 1.

Based on the Chernoff-Hoeffding bound,[32] we have

$$Pr(S_n \leqslant (1 - \delta) \cdot E[S_n]) \leqslant e^{-\delta^2 \cdot E[S_n]/2}. \qquad (4)$$

Matching this formula with our filtering condition of $Pr(S_n \leqslant k) \leqslant 1 - \phi$, we get

$$k = n \cdot \tau_s/N - \sqrt{-2 \cdot n \cdot \tau_s/N \cdot \log(1 - \phi)}. \qquad (5)$$

To implement probabilistic candidate rule filtering, we replace the map from Algorithm 3 with an ordered tree using the occurrence count for ordering the candidate rules. Then after processing each anchor event, we iterate over this tree and remove any candidate rules whose occurrence count is less than or equal to k. Note that k is updated after processing each candidate rule.

In this section, we have defined our terminology, *Alarms*, *Rules*, and *Rule Mining* and given our methodology. First, we have formally defined what an alarm and a rule mean, as well as what the components and requirements of an alarm and a rule are. Second, we formally defined our rule mining algorithm, in a step-by-step manner. Third, we described the *Score* algorithm, which calculates each alarm's influence score. Finally, a probabilistic filtering technique for reducing the system resources used for mining rules was proposed, to make the mining algorithm fast.

4 | ALACA: SCALABLE AND REAL-TIME ALARM DATA ANALYSIS

Alarm Collector and Analyzer is a platform to process vast amounts of alarms that arrive from multiple heterogeneous

sources in real time and identify the root causes of problems where a demo version is demonstrated in our recent paper of Zeydan et al.[25] It is composed of 3 main modules: (1) *Data Collection, Pre-prosessing, and Storage*; (2) *Root Cause Analysis*; and (3) *Real-time Alarm Data Processing* that includes Apache Kafka and Storm for CEP. The implementation integrates the proposed solutions in this paper with open-source big data analytics software. Figure 2 demonstrates the general architecture of our solution.

The *Data Collection, Pre-prosessing, and Storage* module of Figure 2 collects data from heterogeneous sources and transforms them into a suitable format for further analysis. It also persists these data into historical storage. The *Root Cause Analysis* module executes the proposed scalable rule and graph mining algorithm for root cause analysis on alarm events. The *Data Collection* module feeds the Root Cause Analysis module with the required alarm data. The rules discovered by the Root Cause Analysis module are registered into the *Real-time Alarm Data Processing* module. This latter module processes the real-time alarm events within a window to check for possible matches with the previously registered rules. By matching the alarm sequences against rules, we can predict future important alarms in the sequence. The system can also visualize the highly ranked alarms and their relationships via the ALACA Web client. While the first module involves more technical and engineering challenges, the other 2 modules (ie, Root Cause Analysis and Real-Time Event Processing modules) include several contributions in scalable alarm and rule analytics and visualization.

4.1 | Data Collection, Pre-Processing, and Storage

For representing real-time alarms, we use *Event Data Type*. It includes the fields such as Node Name, Alarm Name, Event Time, and Severity as shown in Table 2. For alarm data processing, *Alert Data Type* models the alarm warnings sent back into user in real time. This alert type includes alarm information that can happen in the future. Its contents are listed in Table 3. *Rule Data Type* includes rules that are precalculated and preregistered into the ALACA's CEP module. Using this rule model, warnings or alarm alerts are sent back to the users. The details are in Table 4.

The alarm information is collected and stored in the network operations center. The kind of network status information stored and its format depends on many factors, such as configuration, type, and vendor of the nodes and network equipment. As depicted in Figure 2, the Data Collection, Pre-Processing and Storage module is composed of the following 3 components:

- *Alarm Event Loader*: Loads raw alarm events into Hadoop Distributed File System (HDFS).
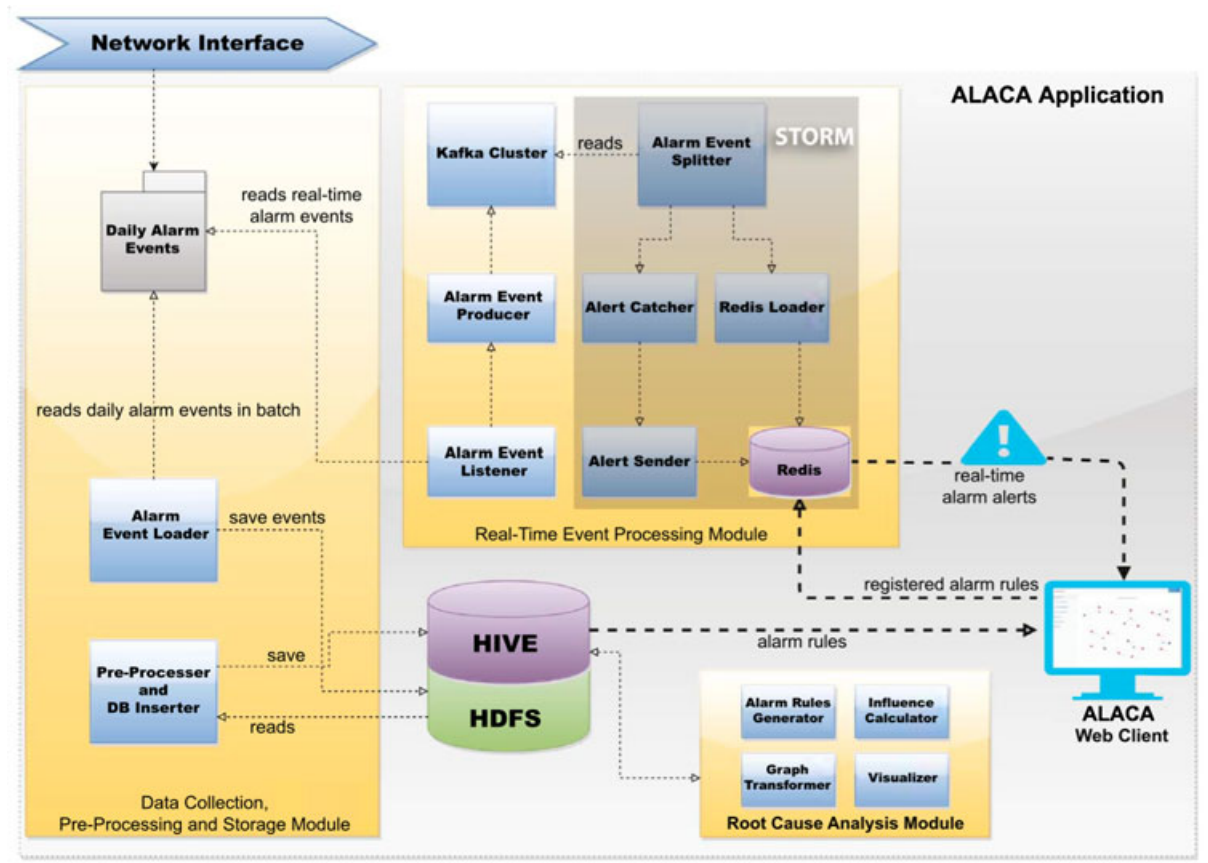
FIGURE 2 High level architecture of ALACA platform

TABLE 3 Alarm *Alert* data format

Field	Explanation
Node Name	Network equipment producing the alarm
Alarm Names	List of possible alarms that can occur
Time Difference	Calculated time interval between the occurring alarms
Captured Time	Required time for the alert data model to be formed in the system
Estimated Occurrence Time	The information about alarms generation time

- *Pre-processor*: Retrieves raw alarm events from HDFS and performs filtering, cleaning, and parsing.
- *Database Inserter*: Stores the cleansed data for permanent storage.

4.2 | Root Cause Analysis

The frequently occurring alarm rules are transformed into a graph representation, which is further used in the system for root cause analysis and visualization. These rules are identified by a 4-phase algorithm: *alarm cleansing*, *alarm indexing*, (ie, Algorithm 1), *rule generation*, and *rule elimination* (ie, Algorithm 2). In summary, in the *cleansing phase*, alarms

TABLE 4 Alarm *Rule* data format.

Field	Explanation
Node Name	Network equipment owning the rule
Antecedent	Ordered set of alarms on the Left-Hand Side
Consequent	Alarm on the Right-Hand Side
Time Duration	Time duration of the rule
Confidence	Confidence value of the rule
Support	Support value of the rule

with matching kinds whose time intervals overlap are merged. In the *indexing phase*, an *interval index* is built over the alarms, such that the list of alarms that are active at a given time can be found quickly (in logarithmic time). In the *rule generation phase*, candidate alarm rules are generated for each possible consequent alarm. For this purpose, for each alarm *X* found in the alarm sequence, the set of active alarms when *X* has started is found using the interval index. Then, candidate rules are created with *X* on the consequent and subsets of the found alarms on the antecedent. Occurrence counters are maintained for these candidate rules across all *X* appearances. After all *X*s are processed, the *elimination phase* starts. The candidate rules with low support or confidence values are eliminated. Here, the support value is the minimum occurrence count for a rule to be retained. And the confidence

value is the prediction accuracy the rule has (how frequent the consequent holds when the precedent does).

4.3 | Real-Time Alarm Data Processing

The Real-time Alarm Data Processing module consists of the following components:

- *Alarm Event Listener*, which reads real-time alarm data from the source and prepares the alarm data for Log Producer's usage.
- *Alarm Event Producer*, which writes alarm data into a Kafka cluster as a producer, by transferring alarm information through a predetermined topic called *Alarm Event*.
- *Kafka Cluster*, which is responsible for retention and alarm log data distribution.
- *Alarm Event Splitter*, which reads alarm data from the Kafka cluster using a specialized adapter, enabling data flow from Kafka into Storm. It also cleans the raw data to prepare it for later analysis and builds the *Event* data format from alarm data.
- *Redis Loader*, which writes the alarm data received from AlarmLog Splitter in *Event* data format, into the Redis database. The goal is to assist the users in seeing the real-time alarm flow with low-latency.
- *Alarm Catcher*, which runs preregistered rules over alarm data coming from AlarmLog Splitter in *Event* data format. Registered rules are modeled using the *Rule* data format. This module performs the calculation based on a window of size τ with a slice of α. This means that last τ time units worth of alarm data is processed against the rules every α time units. If after processing the alarms, a future alarm is predicted to occur based on the rules, these warnings are sent in the *Alert* data format to the Alert Sender module for further processing.
- *Alert Sender*, which writes *Alert* data coming from Alert Catcher module into the database. Alert Sender will not write into the database if an alert is already available in the database or if its estimated occurrence time has passed.

ALACA uses *Apache Storm*[33] for data stream processing and *Redis*[34] for in-memory key-value–based data management. *Apache Storm* guarantees to process incoming data on-the-fly. It is also fault-tolerant. *Redis* is an in-memory key-value store and provides high availability. We derive the alarm rules in the Root Cause Analysis module and process the real-time alarms in the CEP Engine module. *Apache Kafka* is an integral part of our implementation, used to handle real-time data feeds.[35] Kafka provides a fault-tolerant publish/subscribe mechanism with high-throughput and low-latency distribution of data. There are 3 important components within Kafka: producer, consumer, and topic. Producer produces its messages over a topic and sends them to the Kafka cluster. The latter is responsi-ble for holding these messages and distributing them. The consumer receives messages from the topics it is subscribed to and consumes them. Kafka cluster is made up of brokers, responsible for proper distribution of messages into relevant consumers.

4.4 | Benefits of ALACA for scalable stream analytics

The benefits of using the ALACA platform has different dimensions. First, the ALACA platform is a highly available and scalable solution for streaming big data. It can collect vast amounts of data with any format and can response to complex queries in an online manner. Second, ALACA is built upon open source frameworks and can work integrated with a mobile operator's Operations Support Systems. Compared to traditional and more proprietary fault management solutions of vendors (such as IBM Maximo, HP TEMIP, etc) that are used for alarm monitoring inside mobile operators, ALACA is built on top of an open-source platform that is maturing constantly, thanks to its evolving community, which makes it cost-effective as well. Additionally, ALACA has independent modules, which make the platform more reliable. These modules can be updated when a new technology is applicable.

We presented the details of the ALACA platform with its 3 main modules: *Data Collection, Pre-Processing and Storage*, *Root Cause Analysis*, and *Real-Time Event Processing*. *Data Collection, Pre-Processing and Storage* module collects alarm events and loads them into HDFS while performing filtering, cleaning, and parsing of raw data from HDFS and stores the cleansed data. *Root Cause Analysis* module creates alarm rules, transforms them into a graph representation, calculates influence scores from graphs, and visualizes those graphs. *Real-Time Event Processing* module reads real-time alarm data from the sources and prepares the alarm data for the *Data Collection, Pre-Processing and Storage* module, CEP, and the ALACA Web client.

5 | EXPERIMENTS

In this section, we describe our data set, the experiments, and the performance results of the proposed solution. We firstly compare the proposed solution with a traditional sequence mining algorithm. After this, we use 3 main parameters for our evaluation. The first one is the confidence (τ_c) as given in Equation 3. The second is the support (τ_s) as given in Equation 2. The third is the filter confidence (k), as given in Equation 5. We aim to see the impact of these parameter values on rule counts, running time, and memory usage. We run all the experiments 3 times and report the average values as results. Finally, we show the root cause analysis results.

5.1 | Experimental setup

Our data set is obtained from a major telecom provider in 2012. The data have a total of 2 054 296 preprocessed alarm events, which are produced by 80 027 different nodes. For our experiments, we split the data set into 250K, 500K, 750K, 1M, 1.25M, 1.5M, 1.75M, and 2M instances by ordering them by event time. We run our experiments on a workstation, which has 128 GB of memory and 2×2.80 GHz Intel Xeon processor containing 20 cores. We use Java 7 SE to process data. First, we attempted to measure memory usage for each experiment in terms of the number of bytes. However, because of the garbage collection of the JVM, memory measurements did not produce healthy results. Instead, we used the maximum candidate rule map size as given in Algorithm 3 for capturing the memory usage. For each experiment, we reported the running time as maximum, minimum, and average over different nodes.

5.2 | Comparison of proposed and RuleGrowth—sequential rule mining algorithm

Before presenting our experiments on the impact of major workload and system parameters, we compare our proposed algorithm with a traditional sequential data mining algorithm, RuleGrowth.[36] For this evaluation, we implemented a prediction-occurrence model with the alarm rules as we define the formation in Section 3. This model is based on the rule occurrences on the transactions. If the antecedent alarms of the sequential rule of a node occurs in a transaction, then the prediction "the consequent alarm of this rule will also occur in this transaction" is made. If this consequent one also occurs, the prediction is correct. Then, the correctness of the prediction is calculated as *accuracy*. We mine rules from our data set (alarm count: 369 345), which are dated from 10/08/2012 to 23/08/2012 based on the

TABLE 5 Results of prediction-rule model for sequential rules and the proposed Algorithm 3

Algorithm	Prediction count	Rule occurrence count	Accuracy	Recall
RuleGrowth	2702	864	32.0%	69.8%
Proposed method	366	266	72.7%	86.4%

(A) Average Time Usage

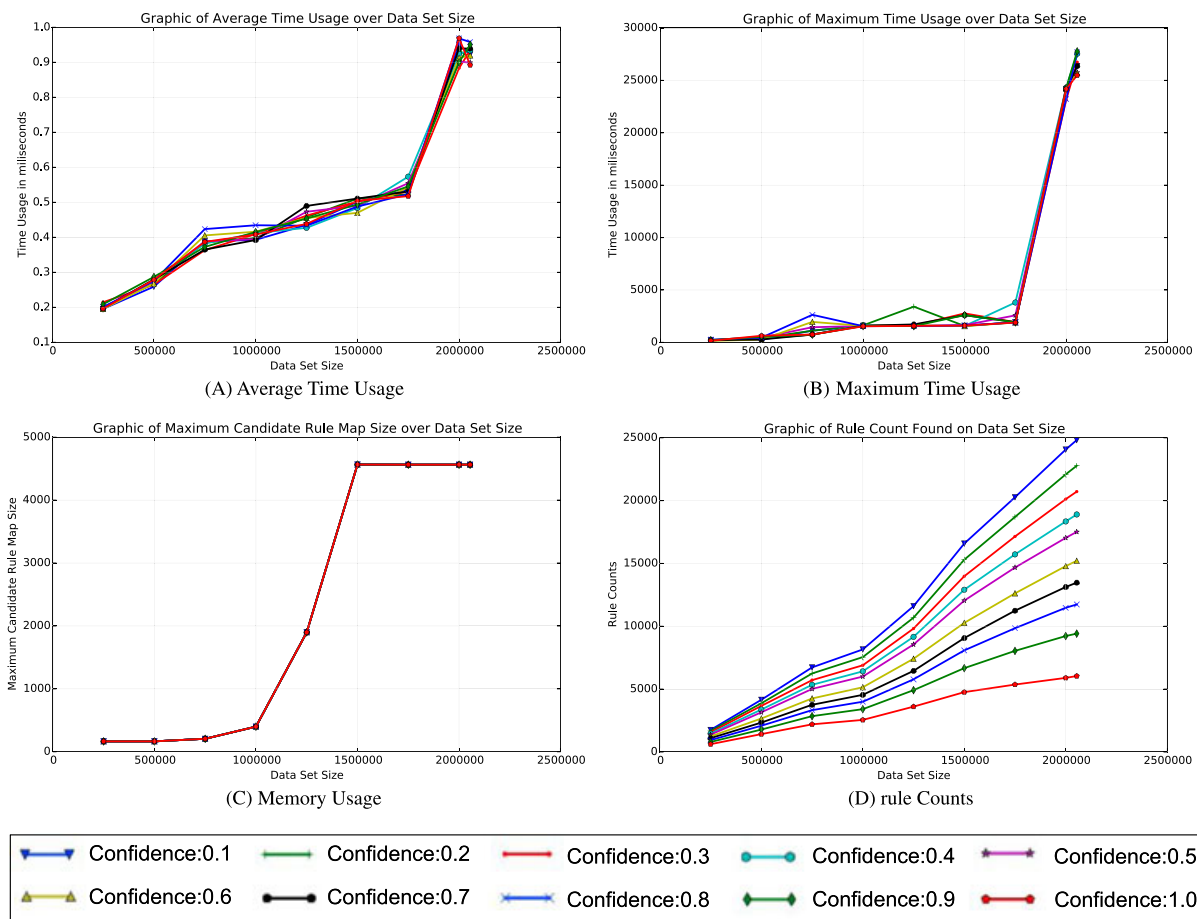(B) Maximum Time Usage

(C) Memory Usage

(D) rule Counts

FIGURE 3 Graphics of different confidence values over data set size
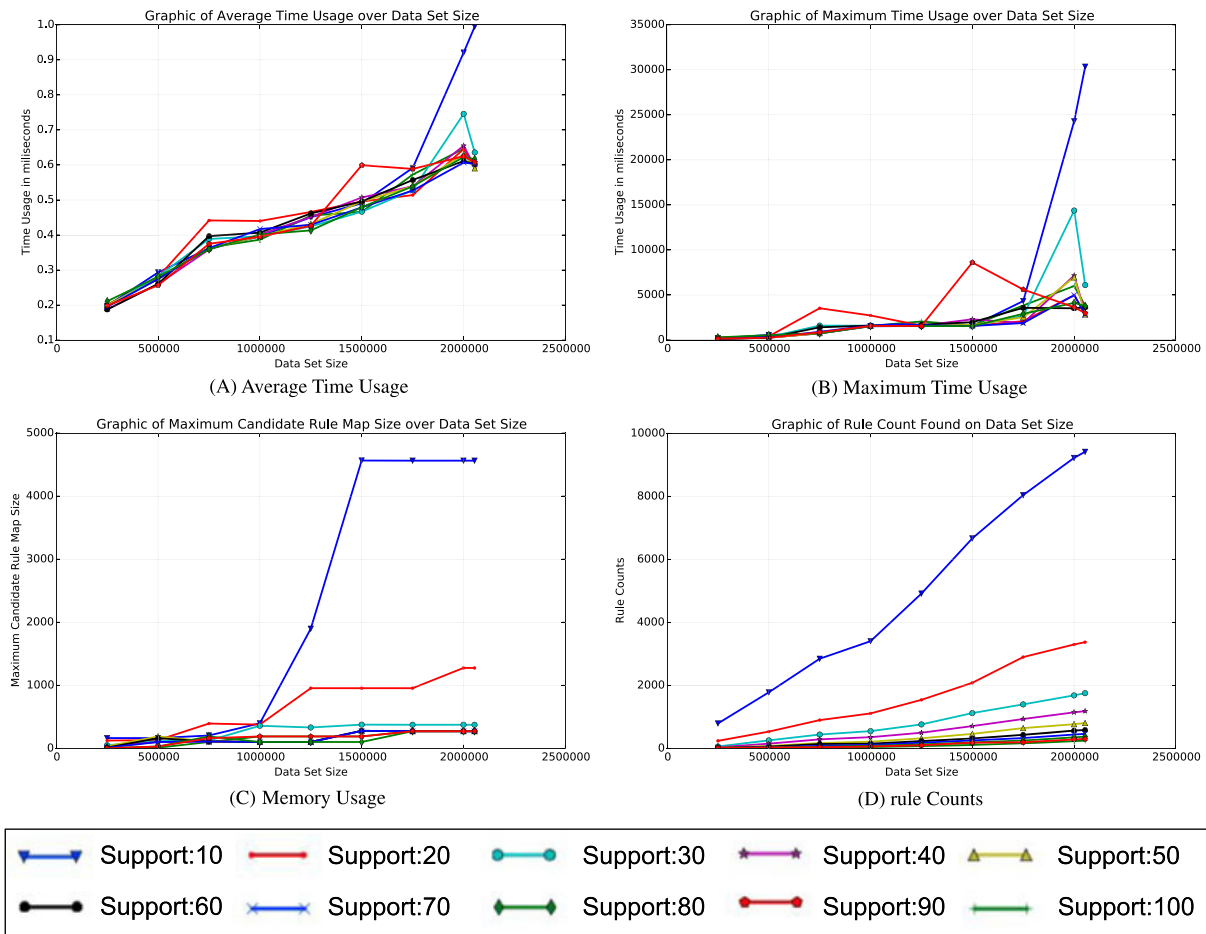
FIGURE 4 Graphics of different support values over data set size

Nokias radio access network logs with support value of 10 and confidence value of 0.9. We create prediction accuracy from these logs (alarm count: 27 855) on 24/08/2012. As shown in Table 5, in our experiments, RuleGrowth algorithm makes 2702 predictions, and 864 rule occurrences take place based on these predictions. Therefore, its accuracy is around 32.0%. Our algorithm makes 366 predictions where 266 of them has occurred. Hence, the accuracy becomes 72.7%. In our experiments, the RuleGrowth algorithm made many predictions while it consumes a lot of resources and time. These results indicate that the proposed method makes more accurate predictions by using less resources and time.

The results for the current experiment are given in Table 5. *Prediction Count* is the number of the antecedent alarms of a sequential rule occurrences on the transactions, and *Rule Occurrence Count* is the number of the consequent alarms of the rule following the antecedents occurrences on the transactions. *Accuracy* is defined as the rule occurrence counts over prediction counts for transactions to measure how correct is the prediction. *Recall* value is defined as the ratio of rule occurrence counts, which are calculated by considering only training data and both training and test data, respectively. The

proposed solution makes 2.27 times better predictions than RuleGrowth algorithm.

5.3 | Impact of confidence

We make first parameter experiment on confidence value as it changes between 0.1 and 1.0. We use fixed support value of 10 and fixed filter confidence value of 0.9. We measure *rule counts, maximum candidate rule size, time max., min., avg.* over data set size. Since the rules are mined as node based, maximum candidate rule map size as given in Algorithm 3 is used for memory usage and time is observed as maximum, minimum, and average. Figure 3 shows the results of different confidence values over data set size. Data set size is on the *x*-axis of all plots. Time usage in milliseconds is on the *y*-axis of Figure 3A and Figure 3B. Maximum candidate rule map size is given on the *y*-axis of Figure 3C and rule counts are given on the *y*-axis of Figure 3D. Different confidence values are the lines on the plots as given on the legend.

There is a linear relation between data set size and time usage in Figure 3A and Figure 3B for this experiment. However, there is no relation between confidence value and memory in Figure 3C. It is because we first create candidate

rules and calculate the confidence values then finally control if they hold the threshold value. That is the same reason of time usages are almost the same of all confidence levels. In all data sets, the minimum time usage is almost 0, so we did not give the minimum time usage plot. Figure 3D shows the

rule counts created for each data set and each confidence level. It shows how the confidence value works on the each data set size. If the confidence value is maximum level, 1.0, the rule counts are minimum for all data sets, and vice versa. For instance, the rule count is 6041 when the confidence level is

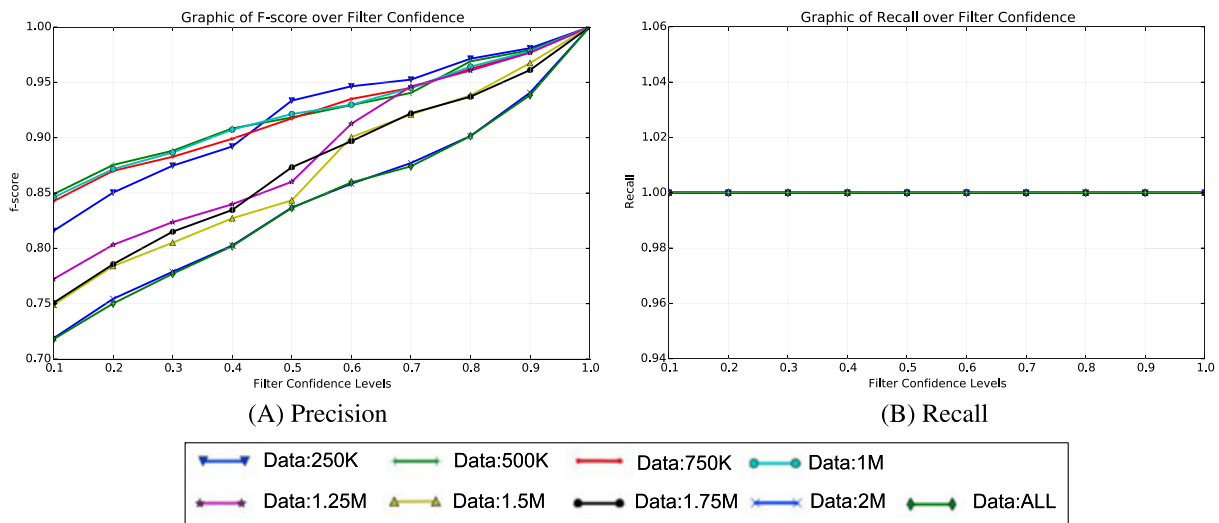**FIGURE 5** Graphics of different filter confidence values over data set size

FIGURE 6 Precision and recall graphics of different filter confidence values over data set size

1.0 for using all data, although the rule count is 24 817 when the confidence level is 0.1.

5.4 | Impact of support

Our next experiment is on support value as it changes between 10 to 100. We use fixed confidence and filter confidence values of 0.9. We measure rule counts, maximum candidate rule size, time *max.*, *min.*, *avg.* as well. Figure 4 shows the results of different confidence values over data set size. Data set size is on the *x*-axis of all plots again. Time usage in milliseconds is on the *y*-axis of Figure 4A and Figure 4B. Maximum candidate rule map size is given on the *y*-axis of Figure 4C and rule counts are given on the *y*-axis of Figure 4D. Different support values are the lines on the plots as given on the legend.

We see that there is a linear relation between data set size and time usage in Figure 4A and Figure 4B as well as previous experiment. Additionally, Figure 4C shows that there is a opposite relation between support value and memory usage. While the support value increases, the memory usage decreases. We see the dramatic change on the memory between support values 10 and 20. For the data set size 1.25M, the candidate rule map size is 1899 for the support value 10 and 956 for the support value 20. For huger data sets, this

TABLE 6 Influence score of alarms in the node DI2146.

Alarm Name	Score
BCF OPERATION DEGRADED	1.00
BCF FAULTY	1.50
ACDC FAULT	0.00
DC POWER MAJOR	1.00
DC POWER MINOR	1.50
HIGH TEMPERATURE	1.75

change is getting bigger. For the data set size 2M, the candidate rule map size is 4564 for the support value 10 and 1277 for the support value 20. We see the rule counts in Figure 4D. There is a reverse parabolic relation between rule counts and support values. When the support value increases, the rule counts dramatically decreases. Although there are 9418 rules created for all instances with the support value 10, there are 3375 rules created with the support value 20.

5.5 | Impact of filter confidence

Our next experiment is on filter confidence as it changes between 0.0 to 1.0. We use fixed confidence value of 0.9 and fixed support value of 10. Filter confidence 0.0 assumes
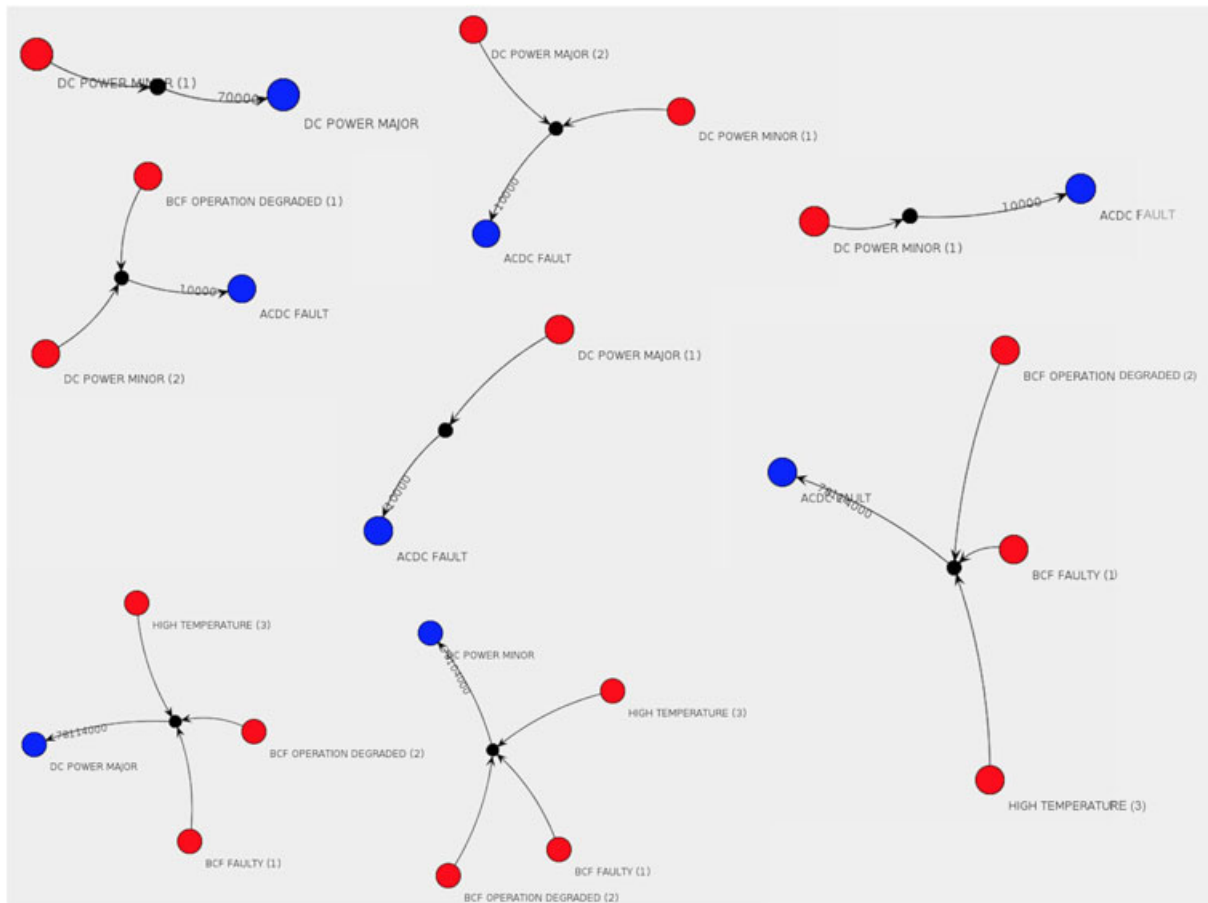

FIGURE 7 The alarm rules of Node Name DI2146's node, which corresponds to a BTS of RAN of MO

that the data set is perfectly uniform, and 1.0 assumes that the data is not uniform at any point. We measure maximum candidate rule size, time *max.*, *min.*, *avg.* as previous experiments. Data set size is on the *x*-axis of all plots of Figure 5. Time usage in milliseconds is on the *y*-axis of Figure 5A and Figure 5B. Maximum candidate rule map size is given on the *y*-axis of Figure 5C and rule counts are given on the *y*-axis of Figure 5D. Different filter confidence values are the lines on the plots as given on the legend.

There is linear relation between data set size and time usage, which can be seen in Figure 5A and Figure 5B as expected. We can also see the filter confidence effect on the time usage by dramatic change between 0.8 and 1.0 in these figures. The maximum time is 16 945 ms when the all data processed with the filter confidence level 0.8. However, the maximum time is 29 488 ms when all the data processed with the filter confidence level 1.0. We can see the same effect on the memory in Figure 5C. While the maximum candidate rule map size is 4564 with filter confidence level 0.9, it is 7392 with filter confidence level 1.0 for all the data. The rule counts over data size for each filter confidence levels are given in Figure 5D. In small data sets, the rule counts can be acceptable at any filter confidence levels. On the other hand, the rule counts are reasonable at filter confidence levels 0.8 and 0.9 for big data sets.

Figure 6 shows precision and recall values of rules. On the *x*-axis, there are filter confidence level values. Data set sizes are the lines for these plots as given on the legend. Precision value is given on the *y*-axis of Figure 6A and recall value is given on the *y*-axis of Figure 6B. Filter confidence levels 0.8 and 0.9 give reasonable precision values for each data set. For instance, when the data set size is 250K, the precision value is 0.944 for the filter confidence level 0.8. For the same filter confidence level, the precision value is 0.821 when all the data are used. Because no different rules are created when filter confidence values change, all recall values are 1.0, which is given in Figure 6B. So the f-values are the same with precision values that is why the f-measure plot is not given.

5.6 │ Alarm impact

To compute the impact score of each alarm, first, we create the alarm rules using our experimental data set described in Section 5.1. For alarm rule generation, we use fixed confidence value of 0.9, fixed support value of 10 and fixed filter confidence value of 0.9. At the end of the process, there are 9418 rules obtained from the data set. Then, based on the constructed rules, we create rule graphs and calculate the corresponding influence scores for each node. For instance, from Node Name *DI*2146, which corresponds to a BTS inside Radio Access Network (RAN), 8 alarm rules are created as shown in Figure 7. The consequent alarm is represented as the blue vertex on an edge originating from a red vertex. To indicate the alert order in the rule, small labels are used to number the antecedent alarms. As a result, the rules can be easily read from the graphs. There are 6 unique alarms used in these rules. The influence scores that
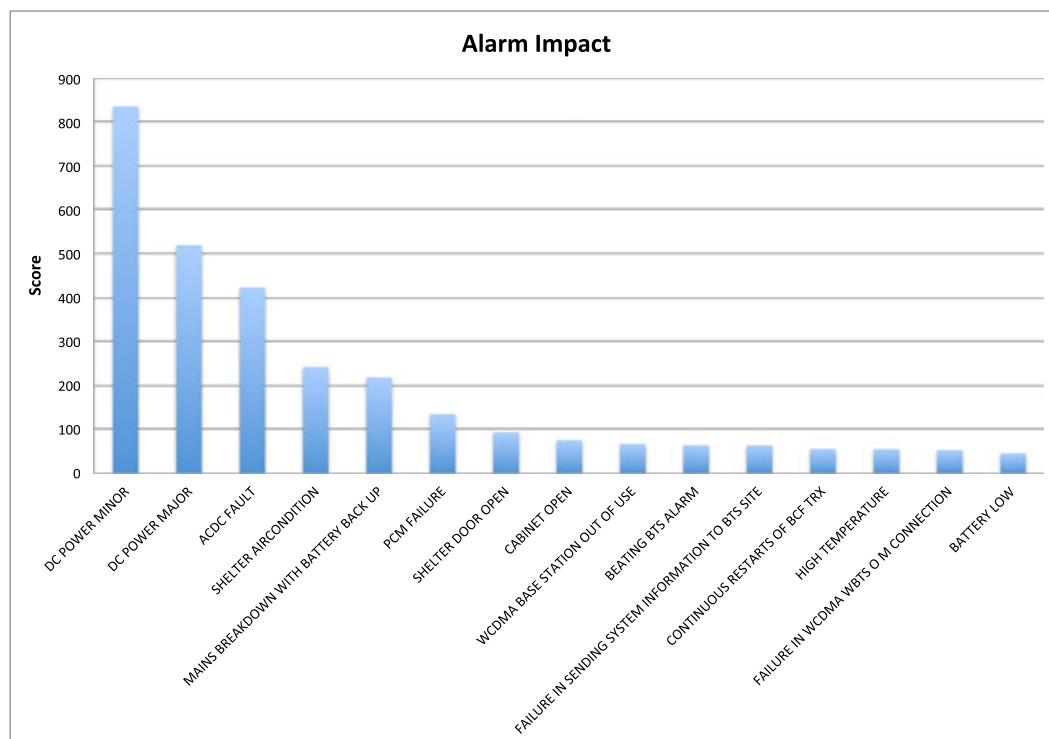

FIGURE 8 The 15 most influential alarms impact scores

are calculated by using Algorithm 6 for these are also listed in Table 6.

We calculate the influence scores of alarms for root cause analyses by using Algorithm 6 for all nodes to identify alarm impact. Because we create our rule graphs node based, the maximum influence score is 1.875 for the alarm *DC Power Minor*. We aggregate the scores to see root causes in Figure 8 as given 15 most influential alarm scores. There are alarm names on the *x*-axis and the aggregated scores on the *y*-axis. The most influential alarms are the alarm *DC Power Minor* with score of 835.125 and the alarm *DC Power Major* with score of 519.500.

To sum up, we run experiments on a real data set collected from a mobile network operator's infrastructure to see the effects of our parameters and observe the most effective alarms that can be the root cause of the subsequent alarms. Although the confidence value does not affect the system resources, it affects the rule counts as expected. The support value has noticeable impact on the system resources used. The rule counts decrease dramatically when the support value increases. We made a final experiment on the filter confidence parameter to see the effect of our sliding window–based approach. When the filter confidence level is 0.8, the system resources are saved and the precision level is higher than 0.80 for all data sets, which is quite reasonable. We see the alarm *DC Power Minor* is more influential than the alarm *DC Power Major* and *ACDC Fault*, which coincides with the fact that before a major power shut down occurs at a given BTS, minor warnings arrive into network operation center of the mobile operator.

6 | CONCLUSION

This paper introduced a real-time alarm data analytics platform developed for a major mobile service provider. The most critical modules of the platform are the real-time alarm capture and the root cause analysis modules. We provided both practical and theoretical insights on large-scale alarm management and mining. Our experiments show that the sliding window–based approach saves system resources by using a filter confidence level of 0.8 or 0.9, while maintaining the high precision of the alarm rules. The developed system is in active use within the network operation center of a major mobile telecom provider. It has started to ease the network operators' job. Therefore, we plan to measure the ticket reductions and the long-term cost savings for the network operator as future work.

This work can be extended in several directions. The current system is developed as node based. As a future work, the system performance can be improved by using clustered nodes and rules can be constructed using these clustered rules. This can result in detecting more influential alarm rules due to

the addition of new rules. One can also extend the root cause analysis to be used for trend analysis of arriving alarms at a seasonal scale.

REFERENCES

1. Yan H, Breslau L, Ge Z, Massey D, Pei D, Yates J. G-rca: a generic root cause analysis platform for service quality management in large ip networks. *IEEE/ACM Trans Networking*. 2012;20(6):1734–1747.

2. Herodotou H, Ding B, Balakrishnan S, Outhred G, Fitter P. Scalable near real-time failure localization of data center networks. *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, New York, New York, USA; 2014:1689–1698.

3. TeMIP Fault Management. 2016. https://en.wikipedia.org/wiki/HP_OpenView, Accessed 30 September 2016.

4. Maximo Asset Management. 2015. http://www-03.ibm.com/software/products/en/maximoassetmanagement/, Accessed 30 September 2016.

5. Rouvellou I, Hart GW. Automatic alarm correlation for fault identification. *INFOCOM'95. Fourteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Bringing Information to People. Proceedings. IEEE*, vol. 2: IEEE, Boston, MA, USA; 1995:553–561.

6. Wallin S. Chasing a definition of alarm. *J Network Syst Manage*. 2009;17(4):457.

7. Duarte EP, Musicante MA, Fernandes HDH. Anemona: a programming language for network monitoring applications. *Int J Network Manage*. 2008;18(4):295–302.

8. Mannila H, Toivonen H, Verkamo AI. Discovery of frequent episodes in event sequences. *Data Mining Knowl Discovery*. 1997;1(3):259–289.

9. Klemettinen M, Mannila H, Toivonen H. Rule discovery in telecommunication alarm data. *J Network Syst Manage*. 1999;7(4):395–423.

10. Wietgrefe H, Tuchs K-D, Jobmann K, Carls G, Fröhlich P, Nejdl W, Steinfeld S. Using neural networks for alarm correlation in cellular phone networks. *International Workshop on Applications of Neural Networks to Telecommunications (IWANNT)*, Citeseer, Stockholm, Sweeden; 1997:248–255.

11. Li T, Li X. Novel alarm correlation analysis system based on association rules mining in telecommunication networks. *Inf Sci*. 2010;180(16):2960–2978.

12. Meira DM. *A model for alarm correlation in telecommunications networks*. Ph.D. Thesis, Federal University of Minas Gerais, 1997.

13. Laxman S, Sastry PS. A survey of temporal data mining. *Sadhana*. 2006;31(2):173–198.

14. Agrawal R, Srikant R. Mining sequential patterns. *Data Engineering, 1995. Proceedings of the Eleventh International Conference on*, IEEE, Taipei, Taiwan; 1995:3–14.

15. Agrawal R, Imieliński T, Swami A. Mining association rules between sets of items in large databases. *ACM SIGMOD Record*, vol. 2: ACM, Washington, D.C., USA; 1993:207–216.

16. Fournier-Viger P, Nkambou R, Tseng VincentS-M. Rule-growth: mining sequential rules common to several sequences by pattern-growth. *Proceedings of the 2011 ACM Symposium on Applied Computing*: ACM, TaiChung, Taiwan; 2011:956–961.

17. Das G, Lin K-I, Mannila H, Renganathan G, Smyth P. Rule discovery from time series. *KDD*, AAAI Press, New York, NY vol. 98; 1998:16–22.

18. Lo D, Khoo S-C, Wong L. Non-redundant sequential rulestheory and algorithm. *Inf Syst*. 2009;34(4):438–453.

19. Yu M, Li W, Li-jin WC. A practical scheme for mpls fault monitoring and alarm correlation in backbone networks. *Comput Networks*. 2006;50(16):3024–3042.

20. Frohlich P, Nejdl W, Jobmann K, Wietgrefe H. Model-based alarm correlation in cellular phone networks. *Proceedings Fifth International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, MASCOTS'97*. IEEE, Haifa, Israel; 1997:197–204.

21. Yemini SA, Kliger S, Mozes E, Yemini Y, Ohsie D. High speed and robust event correlation. *IEEE Commun Mag*. 1996;34(5):82–90.

22. Wallin S, Åhlund C, Nordlander J. Rethinking network management: models, data-mining and self-learning. *Network Operations and Management Symposium (NOMS)*. IEEE, Maui, HI, USA; 2012:880–886.

23. Çelebi ÖF, Zeydan E, Arı İ, İleri Ö, Ergüt S. Alarm sequence rule mining extended with a time confidence parameter. *IEEE International Conference on Data Mining (ICDM)*, St. Petersburg, Russia; 2014:26–35.

24. Sozuer S, Etemoglu C, Zeydan E. A new approach for clustering alarm sequences in mobile operators. *Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP*: IEEE, Istanbul, Turkey; 2016:1055–1060.

25. Zeydan E, Yabas U, Sözüer S, Etemoğlu ÇÖ. Streaming alarm data analytics for mobile service providers. *Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP*: IEEE, Istanbul, Turkey; 2016:1021–1022.

26. Julisch K. Clustering intrusion detection alarms to support root cause analysis. *ACM Trans Inf Syst Secur (TISSEC)*. 2003;6(4):443–471.

27. Zong B, Wu Y, Song J, Singh AK, Cam H, Han J, Yan X. Towards scalable critical alert mining. *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, New York, New York, USA; 2014:1057–1066.

28. Man Y, Chen Z, Chuan J, Song M, Liu N, Liu Y. The study of cross networks alarm correlation based on big data technology. *International Conference on Human Centered Computing*. Springer, Colombo, Sri Lanka; 2016:739–745.

29. Bellec J-H, Kechadi M, et al. Feck: a new efficient clustering algorithm for the events correlation problem in telecommunication networks. *Future Generation Communication and Networking (FGCN 2007)*, vol. 1. IEEE, Jeju, Korea; 2007:469–475.

30. Makanju A, Milios EE, et al. Interactive learning of alert signatures in high performance cluster system logs. *Network Operations and Management Symposium (NOMS), 2012 IEEE*. IEEE, Maui, HI, USA; 2012:52–60.

31. chapter 10.1: Interval Trees, pages 212–217.

32. Mitzenmacher M, Upfal E. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge, United Kingdom: Cambridge University press; 2005.

33. Apache Storm project. 2015. https://storm.apache.org/, Accessed 30 September 2016.

34. Redis database. 2015. http://redis.io/ Accessed 30 September 2016.

35. Apache Kafka. 2015. http://kafka.apache.org/ Accessed 30 September 2016.

36. Fournier-Viger P, Nkambou R, Tseng VS-M. Rulegrowth: mining sequential rules common to several sequences by pattern-growth. *Proceedings of the 2011 ACM Symposium on Applied Computing*. ACM, TaiChung, Taiwan; 2011:956–961.

Selçuk Emre Solmaz is a software engineer at a state agency in Turkey. He received his M.S. degree from Bilkent University in 2017. His research interests are in big data technologies.

Buğra Gedik is an associate professor in the Department of Computer Engineering, Bilkent University, Turkey. He obtained a PhD degree in Computer Science from Georgia Tech, USA. His research interests are in distributed data intensive systems, with a particular focus on stream computing and big data technologies.

Hakan Ferhatosmanoğlu is a professor at Bilkent University, Turkey. He was with The Ohio State University before joining Bilkent. His current research interests include scalable management and mining of multi-dimensional data. He received Career awards from the U.S. Department of Energy, U.S. National Science Foundation, and Turkish Academy of Sciences. He received his PhD degree in Computer Science from University of California, Santa Barbara in 2001.

Selçuk Sözüer is a senior software engineer at OBSS in Turkey. He is currently a master student at Özyegin University. His main research interests are in big data and machine learning.

Engin Zeydan is currently working at Turk Telekom Labs as an innovation and applied research engineer since 2016. Previously, he worked at Avea Iletisim Hizmetleri A.S. as an R&D engineer from 2011 to 2016. He received his PhD degree on December 2010 from the Department of Electrical and Computer Engineering at Stevens Institute of Technology. His research interests are in wireless networks, IP networking, and big data in network management systems.

Çağrı Özgenç Etemoğlu is the manager of the Big Data Software Development group at Turk Telekom. He holds a PhD degree from the University of California Santa Barbara, USA. His research interests are in big data systems.