Mining causes of network events in log data with causal inference

Satoru Kobayashi
University of Tokyo
sat@hongo.wide.ad.jp

Kensuke Fukuda
NII/Sokendai
kensuke@nii.ac.jp

Hiroshi Esaki
University of Tokyo
hiroshi@wide.ad.jp

***Abstract*—Network log message (e.g., syslog) is valuable information to detect unexpected or anomalous behavior in a large scale network. However, pinpointing failures and their causes is not an easy problem because of a huge amount of system log data in daily operation. In this study, we propose a method extracting failures and their causes from network syslog data. The main idea of the method relies on causal inference that reconstructs causality of network events from a set of the time series of events. Causal inference allows us to reduce the number of correlated events by chance, thus it outputs more plausible causal events than a traditional cross-correlation based approach. We apply our method to 15 months network syslog data obtained in a nation-wide academic network in Japan. Our method significantly reduces the number of pseudo correlated events compared with the traditional method. Also, through two case studies and comparison with trouble ticket data, we demonstrate the effectiveness of our method for network operation.**

I. Introduction

Maintaining a large-scale network reliably has been a fundamental requirement in network management. It is however not an easy task in reality because of highly distributed, ever-evolving, and heterogeneous nature of operational networks [1]. One of the effective ways to track network status is to deploy monitoring agents in the network and collect log information corresponding to a change of status. In operational networks, syslog [2] is widely used for such purpose. The detailed log messages allow us to better understand failures and their causes. Nonetheless, it is usually hard for network operators to identify them because of a large amount of system log data produced by a large set of network devices (e.g., routers, switches, and servers).

To this end, various approaches have been taken for improving network monitoring and diagnosis with log messages. A simple approach is clustering log messages related to a network event (e.g., failure) into a correlated group, and analyzing the group in detail. This assumes that the network event can yield a set of messages from some monitored network functionalists. One of the problems of log analysis is that co-occurrence of log messages does not always mean causal relations. Timestamp of messages is helpful in determining the causality, however, the correctness of the timestamp is an issue for appropriate analysis. Furthermore, appearance of network log messages is discrete and sparse; it makes us difficult to identify causality of events.

In this paper, we intend to extract causal relations beyond co-occurrences in log messages in order to identify important network events and their causes. For this, we leverage on a causal inference algorithm, called PC algorithm [3], [4]. It outputs directed acyclic graphs (DAGs) that connect events with causality from a set of network logs. There are some issues when applying causal inference algorithms to the network logs; (1) A large-scale network is composed of multiple vendor's devices, and various types of messages appeared in the network. (2) Occurrence of messages is discrete and sparse that is not assumed in causal inference algorithms. (3) All of the detected causalities are not necessarily important, in the context of network management.

To overcome these issues, we propose a mining algorithm built on the causal inference. We apply our proposed algorithm to 15 months long syslog messages (34M in total) collected from a nation-wide research and education network in Japan [5]. We obtain a reasonable number of causal edges with low false positive rates, compared with a traditional cross-correlation based method. Furthermore, we design simple heuristics that suppress commonly appeared causality and thus highlight uncommon causal relationships. Through case studies and comparison with trouble ticket data, we demonstrate the effectiveness of our approach.

The contribution of this paper is twofold: First, we introduce an idea of causal inference to highly complicated network log data, i.e., a wide variety of log messages sparsely generated by heterogeneous network devices. Second, our method reports a small number of meaningful network events with causal relations from a huge number of log messages; it is useful for daily network operation.

II. Related Work

Prior literature has been devoted to automated troubleshooting and diagnosis with system logs. Some works conduct contextual analysis of log data to retrieve useful information in troubleshooting. Contextual log analysis can be classified into 4 groups: model, spatial, relational, and co-operative approaches.

Model approaches present system changes behind log events as some models, especially state transition models. Salfner et al. [6] use hidden semi-Markov model for failure prediction. Yamanishi et al. [7] analyze system logs with multidimensional hidden Markov model. Beschastnikh et al. [8] generate finite state machines from execution traces of concurrent systems to provide insights of the system for developers. Fu

978-3-901882-89-0 @2017 IFIP

et al. [9] generate decision trees of system state transition from program logs for distributed debugging. Generally, these approaches enable us to understand system behaviors and to predict failures in the near future.

Spatial approaches present log events in multidimensional space and analyze them with classification techniques. Kimura et al. [10] characterize network faults in terms of log type, time, and network devices with a tensor analysis of system logs. Sipos et al. [11] use multi-instance learning approach to extract system failures from system logs. Fronza et al. [12] predict system failures by classifying log events based on support vector machines. These approaches are especially useful for fault localization.

Relational approaches extract relations of time-series of events. The detected relations are used for further analyses like graph approach, and useful especially for fault diagnosis. Root cause analysis with system logs has also been a hot topic in the context of network management. A popular approach is to infer causal relations among events in system logs. Zheng et al. [13] detect correlated events in system logs in a supercomputer system and remove pseudo correlations with conditional independence. Nagaraj et al. [14] generate dependency networks [15] (similar to Bayesian networks) of events in system logs. These approaches are not effective for sparse data, like system logs of network devices, because they employ a probabilistic method to find conditional independence. Mahimkar et al. [16] take an approach to extract failure causality from correlations. They use multiple regression coefficient, but this approach requires large processing time if the system logs include a large number of events. Some approaches estimate causal relations without causal inference.

There are some other approaches to detect root causes of troubles in system logs. Tak et al. [17] generate reference models, explaining dependencies of events, by a heuristic-based method for cloud system logs. They effectively use domain knowledge of the cloud system, which is usually not available for other applications. Lou et al. [18] estimate causal relations with heuristic-based rules of timestamps and message variables. They largely rely on heuristics of log event dependence, which is difficult to generalize.

Co-operative approaches depend on not only system logs but also other datasets. Yuan et al. [19] pinpoint errors in source code by matching a function call graph with error log messages. Scott et al. [20], [21] troubleshoot SDN control software based on causal relations estimated with external causing tools. These approaches need the external data to be available on the systems.

Our work is categorized into the relational approaches based on causal inference. However, existing works are resource-expensive to estimate causal relations. Closer to our work, Chen et al. [22] generate causality graphs for pinpointing the source of network traffic delay. They employ PC algorithm, a method to estimate DAGs from statistical data based on conditional independence. Their target data is not system logs but some monitored parameters such as RTT and TCP window size. Again, our aim is to propose an efficient causal inference algorithm based on graph approaches for system logs.

III. Methodology

In order to detect root causes of network events in system logs, we intend to infer causal relations among log events obtained at an operational network. In this section, we first provide a basic algorithm, called PC algorithm [3], [4], to infer causal structure from event time series efficiently. Next, we propose our data processing method from a set of log messages to determine the causality of network events. We also describe our dataset gathered in a Japanese academic network [5].

A. Detecting causality

The key idea of our proposal is to detect causality of two given events in network logs. The causality is a partial order relationship different from correlation which is typically quantified by a correlation coefficient. Using correlation as a causality yields many false positives. Existence of a positive correlation between two events does not always imply causality. Checking timestamps of two correlated events is helpful in determining causality. The timestamp of system logs is not always reliable for determining causal directions due to NTP synchronization error, variation of packet latency (jitter), and network failure. Thus, we need to estimate causal directions among events without timestamp, in theory. Such algorithms have been developed in a field of causal inference. In this subsection, we introduce a method called PC algorithm to infer causal relationships from a set of events.

We first introduce a concept of conditional independence. Assume that there are three events A, B, and C. A and B are conditionally independent for given C if

$$P(A, B \mid C) = P(A \mid C)P(B \mid C), \tag{1}$$

where the events A and B are independent as long as C appears. If A and B have a causality with C, A and B are independent because they always occur with C. In other words, a correlation between A and B disappeared by considering the related event C. Note that C can represent multiple events.

PC algorithm [3], [4] is a graph-based method to reconstruct causal relationships among nodes with conditional independence. It assumes a direct acyclic graph (DAG) of events corresponding to the causality of events; it does not allow any loops. PC algorithm consists of four steps (see also Figure 1).

1) Construct a complete (i.e., fully-connected) undirected graph from nodes (events).
2) Detect and remove edges without causality by checking conditional independence.
3) Determine edge direction based on V-structure.
4) Determine edge direction with orientation rule.

For testing conditional independence in step 2, we first cut edges (say $A - B$) whose nodes are conditional independence without another node., i.e., this is a special case of Eq. 1 similar to the usual correlation coefficient. We then check the remaining edges ($A - B$) by conditional independence with additional nodes (C) that connect to the two nodes (A and B) using Eq. 1. We remove the edge if at least one of the

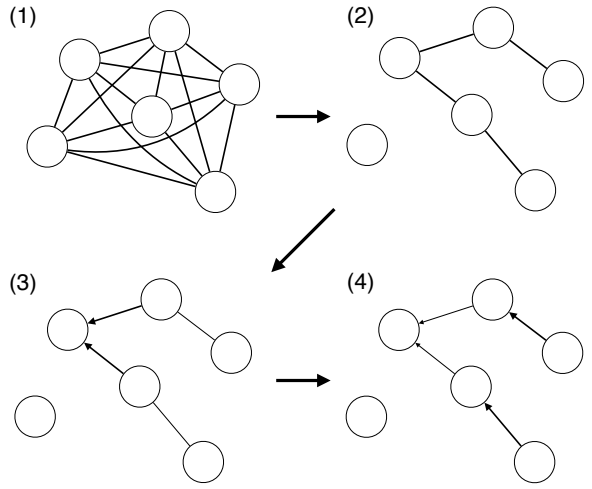

Fig. 1. DAG estimation process in PC algorithm

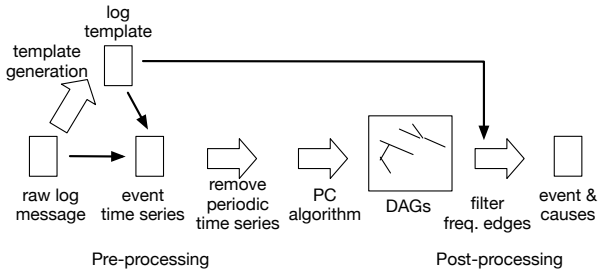

Fig. 2. Processing flow

other nodes holds the conditional independence. In step 3, V-structure is a condition to decide on a direction of edge. Let three nodes (events) U, V, W be a part of a graph $U-V-W$; U and V are correlated and V and W are correlated. One obtains a causal relationship $U \rightarrow V \leftarrow W$ if U and W are not conditionally independent for V. This V-structure is a rule to infer causality (arrow) in an undirected graph. In step 4, Orientation rule is a rule to prevent making a loop among events by the definition of DAG [23]. Some edges can be undirected even after applying PC algorithm if one does not have enough information to decide edge directions.

B. Algorithm

This section provides the processing flow of our log mining algorithm (see also Figure 2).

1) Pre-processing: As an input of PC algorithm, we construct a set of time series corresponding to each event from the original log messages obtained from one network device. First, we extract log templates from the original log messages. The log template is a log format whose variables (e.g., timestamp, IP address, port) are replaced by wild card (*); thus it is a standard (i.e., comparative) format of log messages. There are many log template generation algorithms from log messages [24]–[26]. Here, employing a supervised learning-based log template generation algorithm [27], we extracted 1414 templates from 35M log messages in our dataset (see also § III-C). Examples of log templates are shown in Figure 7 and Figure 9.

Next, we construct a set of event time series generated by each log template per device (router or switch) from the all log data. In other words, each time series contains the number of appearances of one log template from one device in a time bin (size b). We then remove a large number of unrelated event time series indicating strong temporal periodicity. Such a periodic log template represents daily regular events and would be a source of misdetection of the causality. To track the periodicity, we simply calculate the self-correlation coefficient with lag τ for the event time series ($x(t)$):

$$\rho(\tau) = \frac{\sum_t (x(t) - \bar{x})(x(t+\tau) - \bar{x})}{\sqrt{\sum_t (x(t) - \bar{x})^2}\sqrt{\sum_t (x(t+\tau) - \bar{x})^2}}. \quad (2)$$

We identify the maximum value of $\rho(\tau)$ for $\tau > 0$, and we discard the periodic event time series with $\rho(\tau) > 0.9$.

2) Calculation of conditional independence: Furthermore, we apply PC algorithm in § III-A to a set of event time series to generate causality graphs. For evaluating the conditional independence of nodes X and Y with another node Z, we perform a statistical test called G-square test [28], a natural extension of Chi-square test. The G-square statistic is defined as:

$$G^2 = 2mCE(X, Y \mid Z), \quad (3)$$

where m is the data length and $CE(X, Y \mid Z)$ is a conditional cross entropy of event time series X, Y, and Z ($x(t)$, $y(t)$, and $z(t)$, respectively). We check a p-value of the null hypothesis of the test with a threshold value of 0.01. It is noted that event time series should be binary time series for calculating the cross entropy, thus we use a binary event time series (consisting of zero and one) generated from the original event time series by converting each value for $x(t) > 1$ to $x(t) = 1$. We discuss the parameter dependency of bin size b for the edge detection in § IV-A.

3) Post-processing: The output of PC algorithm is a set of causalities. However, PC algorithm does not provide any information on the importance of events. Thus, we require a further step to filter commonly detected (or uninterested) causality. In this study, we simply count the number of appearances of frequently appeared edges. This happens because same edges appear in many devices over time. Thus, we remove frequently appeared edges with a threshold for easily identifying unusual important causality. We discuss the effect of the post-processing in § V-C.

C. Dataset

To evaluate the effectiveness of our approach, we use a set of backbone network logs obtained at a Japanese research and education network (SINET4 [5]) that connects over 800 academic organizations in Japan. The nation-wide network consists of eight core routers, 50 edge routers, and 100 layer-2 switches composed of multiple vendors. A centralized database stores all syslog messages generated by the network

TABLE I
CLASSIFICATION OF LOG MESSAGES

Type	#messages	#processed	#templates
System	3,012,500	994,125	801
Network	123,988	123,988	119
Access	5,489,262	6,792	88
Operation	15,737,987	8,275	70
Cron	10,799,133	20	9
NTP	224,654	224,654	21
SNMP	74,029	74,029	45
BGP	28,017	28,017	55
MPLS	18,524	18,524	85
OSPF	4,184	4,184	11
Other	847	847	110
Total	35,513,125	1,483,455	1,414

devices, though some of the messages can be lost in the case of link failures. Each message contains additional information such as timestamp and source device name (or IP address) based on syslog protocol. We analyze 456 day-long consecutive logs composed of 35M log messages in 2012-2013.

To easily understand log messages for pinpointing events and its root causes, we manually label event types to the generated log templates as listed in Table I. System logs from network devices are classified into four groups: system, network, access, and operation. The group System shows internal processes of devices. The group Network is linked to protocols and devices for communications. The group Access appears if one intends to communicate with devices for management purposes. The group Operation corresponds to configuration changes by operators. Also, we separately introduce some external groups that are related to frequently used services.

In the table, the column "#messages" represents the number of raw log messages, "#processed" is the number of processed messages (after the pre-processing), and "#template" is the number of identified log templates. We confirm that the group cron is mostly removed by pre-processing, due to periodic behavior of the events. Similarly, the groups Operation, Access, and System are suppressed largely. In total, 4% of the number of log messages are used as an input of PC algorithm. Also, the template generation algorithm outputs $1,414$ log templates.

In addition, we check a set of trouble tickets issued by SINET network operators. This data consists of a date and a summary of an event, though it only covers large network events. We use this data for evaluating the detection capability of our proposed algorithm (§ V-E).

The network consists of a large number of network devices, so we divide the data into eight subsets corresponding to a sub network with one core router, edge routers and switches that connected to the core router. We also analyze one-day-long log data because we target short-term causality instead of long-term one. In other words, we generate 3,648 DAGs (456 days and 8 devices) from the entire dataset.

TABLE II
DEPENDENCY OF BIN SIZE ON THE NUMBER OF EDGES

Bin	Directed edges	(Diff. device)	Undirected edges	(Diff. device)	All edges
300s	1,373	489	4,810	1,120	6,183
180s	1,804	686	5,503	1,259	7,307
60s	2,623	827	5,990	1,022	8,613
30s	3,046	821	6,224	1,132	9,270
10s	3,369	721	5,555	863	8,924

TABLE III
RATIO OF OVERLAPS IN DETECTED EDGE SETS FOR DIFFERENT BINS

Compared bin sizes	Directed edges	(Diff. device)	Undirected edges	(Diff. device)	All edges
300s-180s	0.70	0.44	0.83	0.60	0.80
180s-60s	0.71	0.43	0.80	0.46	0.78
60s-30s	0.79	0.52	0.87	0.62	0.85
30s-10s	0.67	0.36	0.73	0.54	0.71

IV. VALIDATION

A. Bin size dependency

In the previous section, we explained the basic flow of the processing. However, the performances of the algorithm depend on the parameter setting. The most important parameter in the algorithm is the time bin size b for aggregating events. Intuitively, larger b detects more false positive edges in causality graph because unrelated events can be co-located in the same bin. Similarly, smaller b makes it difficult to detect a causality between two events with a certain delay. To investigate the dependency of the bin size b on the number of detected edges, we carefully observe the outputs of PC algorithm for different bin sizes from 10s to 300s.

Table II lists the total number of detected directed and undirected edges from the dataset. The column (Diff. device) represents the edges between two different devices. First of all, we can see that the number of detected edges is smaller for larger b. This result looks differently from our intuition that larger b detects more false positive edges. As described in § III-B, cross entropy affects the result of the conditional independence test. Larger b makes the total number of bins in a time-series smaller because the dataset size is fixed to one day. The small number of bins decreases information gain, which yields a small cross-entropy value. Also, the information on the number of same log templates is not considered in our binary time series. Thus the value of the cross entropy becomes smaller for larger b, resulting more rejections of the statistical test. Note that the reduction of edges with larger b does not mean a higher accuracy of detected edges. Next, we confirm a decrease in the number of detected edges with smaller b. In particular, the decrease in the number of edges between two different devices is clear. This result is consistent with our intuition, because the edges between different devices usually have a larger delay than others because of the network latency.

Table III shows the ratio of overlapped edges between neighboring bin size settings (e.g., 300s vs 180s). The ratio of overlap is defined as $\frac{|E_i \cap E_j|}{\min(|E_i|,|E_j|)}$, where E_i (resp. E_j) is a set of detected edges obtained by bin size i (resp. j).

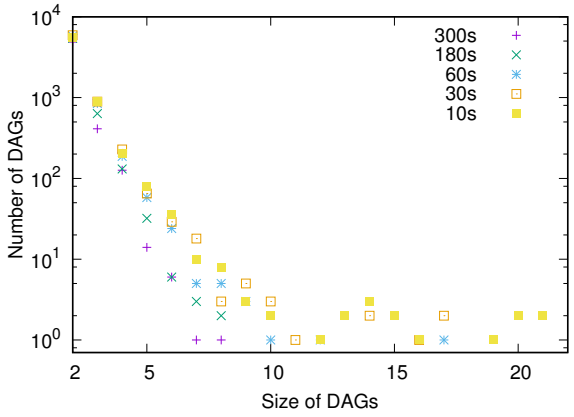

Fig. 3. Distribution of size of DAGs

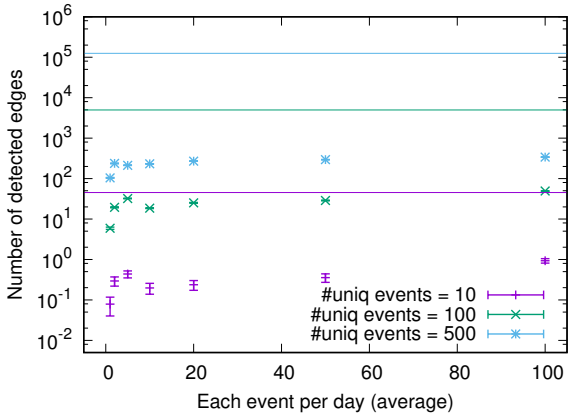

Fig. 4. The number of false positive edges in a Poisson arrival process

Higher overlap indicates less dependency on the detected edges. From the result, we see that any listed combinations of bin sizes in this table have about 20% of differences. this difference indicates that some edges can be detected only for a particular bin size, caused by two events with a certain time lag. Similarly, the ratio between different devices is smaller because associated log messages across different devices usually have a certain time lag.

Next, we examine the size of DAGs produced by PC algorithm. We focus on connected subgraphs, i.e., consisting of at least two nodes. Figure 3 shows the distribution of size of DAGs found in all log data; as expected, most DAGs are small and a few are large. We also confirm that smaller b generates more connected subgraphs. It is consistent with the previous result that a larger number of edges are detected for smaller b. Further investigations on detected large subgraphs identify two typical behaviors: One is an event that causes multiple devices to have similar events. For example, if a NTP server has a failure on its service, all devices output similar log messages about NTP synchronization error. These events appear at the same time, and construct a large subgraph. In this case, the subgraph is reasonable as the causal relations in the system logs. The other behavior is that multiple unrelated events have a common connected event. Such connected event appears frequently and its causality is false positive.

From these analysis results, we conclude that a setting with more edges is helpful in further investigation even if it may contain more false positives. Thus, we empirically employ $b = 60s$ for further analysis. In terms of processing time, it takes about four hours to generate all causality graphs from the 15 months data on a commodity computer.

B. False positives

Here, we investigate a possibility that two events are accidentally connected by an edge in PC algorithm. For synthetic multiple random events without causality, any detected edges are clear false positives by chance.

We prepare 10 (also 100 and 500) unique events and generate these events whose occurrence follows a Poisson process with different arrival rates. Applying our method to these surrogate time series ($b = 60$s), we count the number of detected edges. These edges are considered as false positives. For 10 unique events (also 100 and 500), the maximum number of detected edges among nodes is 45 ($4,950$ and $124,750$), i.e., the worst case.

Figure 4 illustrates the number of falsely detected edges for different event arrival rates. We can confirm that the number of false positives are small and fairly stable over different arrival rates. The false positive ratios are 1.9% for the 10 unique events, 1% for the 100 events, and 0.3% for the 500 events. Thus, we may at least find about 1% of false edges in a given dataset, in principle.

V. Results

In the previous section, we analyzed the parameter dependency of the algorithm on the number of detected causality. This section highlights the effectiveness of the proposed algorithm in the view point of network operation.

A. Detected edges

We first list the classification of detected edges (Table IV) from the pre-processed log messages (Table I) by PC algorithm. This table presents the number of neighbor nodes (i.e., events) for detected edges, because some edges connect two different classification types of events.

Major types are System, Network, NTP and BGP. Events of System and Network are related to external changes, which generate causal relations. NTP events often generate large causal graphs because an NTP server error affects all related network devices. Our causal analysis is beneficial for BGP analysis due to its frequent reporting on BGP state changes.

In addition, we show the number of neighbor nodes of edges between nodes belonging to the same type. Most of edges in BGP and MPLS are closed inside own type (i.e., inner-types). The processes of network protocols are usually independent of other processes in normal operation, and synchronized to other ends. Most edges in Operation are also closed inside own type (i.e., inner-types). This is mainly because a change

TABLE IV
CLASSIFICATION OF NEIGHBOR EVENTS OF DETECTED EDGES: THE PERCENTAGE SHOWS THE RATIO TO ALL EDGES. THE COLUMN OF DIRECTED SHOWS ONLY THE NEIGHBOR EVENTS OF DIRECTED EDGES.

Type	All Edges	(Directed)	Edges of Inner-types	(Directed)	Important Edges	(Directed)
System	5,058	1,153	3,074 (61%)	558	1,202 (24%)	436
Network	2,530	592	2,026 (80%)	380	491 (19%)	193
Access	911	160	602 (66%)	54	218 (24%)	57
Operation	1,529	233	1,320 (86%)	146	184 (12%)	72
NTP	3,018	1,108	1,312 (43%)	594	254 (8%)	132
SNMP	295	94	60 (20%)	22	176 (60%)	67
BGP	2,660	1,141	2,370 (89%)	1,290	314 (11%)	170
MPLS	1,189	459	1,142 (96%)	434	221 (19%)	81
OSPF	31	2	18 (58%)	0	31 (100%)	2
Other	5	4	0 (0%)	0	5 (100%)	4
Total	17,226	4,964	11,924 (69%)	3,478	3,096 (20%)	1,232

of configuration produces multiple log messages; These events are classified as Operation type, and their edges are the majority of this type. In contrast, most edges in SNMP are connected to other types (i.e., inter-types). SNMP events appear mainly with events of System types, which is consistent with the role of SNMP.

We also show the number of neighbor nodes of edges that is estimated as important information for operators in post-processing step. SNMP events are likely to provide meaningful information than other types. This is not against our intuition, because SNMP events appear with some unexpected behaviors of devices.

B. Comparison with cross-correlation method

Here, we compare the performance of our method to a conventional correlation-based method. Cross-correlation is the most popular method to analyze system logs in terms of co-occurrence. Let two events X and Y appear $x(t)$ and $y(t)$ in time t. Cross-correlation ρ is defined as:

$$\rho = \frac{\sum_t (x(t) - \bar{x})(y(t) - \bar{y})}{\sqrt{\sum_t (x(t) - \bar{x})^2}\sqrt{\sum_t (y(t) - \bar{y})^2}}. \quad (4)$$

If ρ close to 1, X and Y are positively correlated; but X and Y are not correlated for $\rho = 0$. We judge the co-occurrence between two events with a threshold ρ_t for ρ.

We compare the number of detected edges in the proposed PC algorithm to that in the cross-correlation based method with different thresholds ρ_t. Figure 5 shows a sensitivity analysis of the cross-correlation method on the number of detected edges. We use 60s bin size for constructing event time series. We observe that the number of detected edges in the cross-correlation method decreases for high ρ_t. However, the number of detected edges in the cross-correlation method is 20 times larger than ours. Thus, the simple correlation-based method cannot remove suspicious correlation, which results in more edges; If some events appear at the same time, the cross-correlation generates a complete graph. On the other hand, our approach detects more reasonable edges thanks to conditional independence.

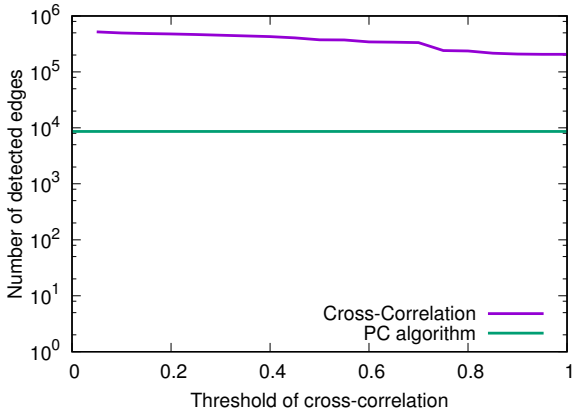

Fig. 5. The number of detected edges for the proposed algorithm and cross-correlation algorithm

C. Post-processing

As describe in § III-B3, the detected edges contain no information on importance of the causality which may help network operators. Some edges can appear frequently in the dataset for two reasons: one on different days and the other on different pair of network devices. Figure 6 represents the rank of edges and the number of such edges appeared. The number of edges produced by PC algorithm is $8,613$ edges (= 19 edges/day). We see that the top 10% of edges accounted for 80% of the number of edge appearances. Manually observing such edges, we confirm that they are mostly edges showing a causality between an input command and its acknowledgement. Thus, we filter them (with a threshold; 10%) and concentrate on the rest of them ($1,722$ edges = 3.8 edges/day).

D. Case studies

To show the effectiveness of the proposed algorithm, we provide two case studies.

1) Broadcast storm: The first case is regarding broadcast storms in some period. During this period, we find two broadcast storm related log templates; one for detection, the other for recovery. We also notice SSH login failure events triggered by a monitoring script in the storms. The monitoring

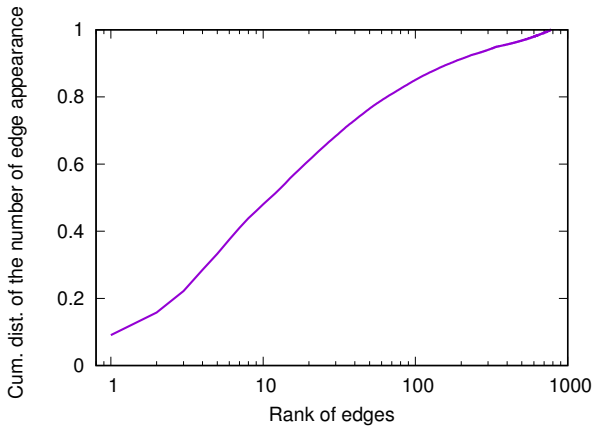

Fig. 6. Rank of edges and their number of appearances

```
650 (EVT ** ** ** ACCESS ** ** Login incorrect **.)
685 (EVT ** ** ** PORT ** ** ** NIF ** Port ** storm detected.)
684 (EVT ** ** ** PORT ** ** ** NIF ** Port ** storm recovered.
```

Fig. 7. Detected log templates (broadcast storm)

script fails to re-establish a SSH connection to a target device due to the storm. The detected log templates by the proposed algorithm are shown in Figure 7. The first one (Template ID: 650) is the failure of SSH login, and the rest of two (ID: 684 and 685) are related to broadcast storm events. In our data, we observe four events on SSH login failure and a series of events repeatedly detecting and recovering broadcast storm. These are all found on the same device.

Figure 8 shows detected causality graphs and its manually inferred plausible causality (a) (ground truth). PC algorithm with 60s bin generates a DAG shown in (b). Compared to the ground truth, PC algorithm reasonably detects causal relations except causal directions.

Visually checking DAGs for different bin size, we confirm that the SSH event (a dotted rectangle) is missed for a smaller bin size ($b < 60s$) as shown in (c). This is due to a time lag between the SSH event and the detection of a broadcast storm.

Furthermore, we investigate the result achieved by the cross-correlation method. There are no nodes appeared for a high threshold ($\rho_t > 0.7$). The three nodes appeared similar to (b) in a limited range of the threshold ($0.1 < \rho_t < 0.2$) but no direction. Otherwise, we observe the case (c) or a fully

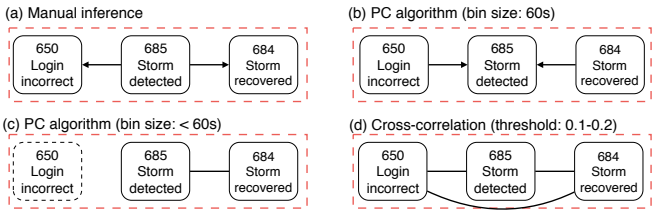

Fig. 8. Ground truth and detected causalities (broadcast storm)

```
679 (EVT ** ** ** PORT ** ** ** Port up.)
687 (EVT ** ** ** PORT ** ** ** Error detected on the port.)
850 (RPD_BGP_NEIGHBOR_STATE_CHANGED: BGP peer ** (** AS **)
   changed state from ** to ** (event **))
1134 (bgp_hold_timeout:**: NOTIFICATION sent to ** (** AS **):
   code ** (Hold Timer Expired Error), Reason: holdtime expired
   for ** (** AS **), socket buffer sndcc: ** rcvcc: ** TCP
   state: **, snd_una: ** snd_nxt: ** snd_wnd: **
   rcv_nxt: ** rcv_adv: **, hold timer **)
1173 (bgp_send: sending ** bytes to ** (** AS **) blocked
   (no spooling requested): Resource temporarily unavailable)
1274 (last message repeated ** times)
91 (** (** AS **): reseting pending active connection)
```

Fig. 9. Detected log templates (BGP)

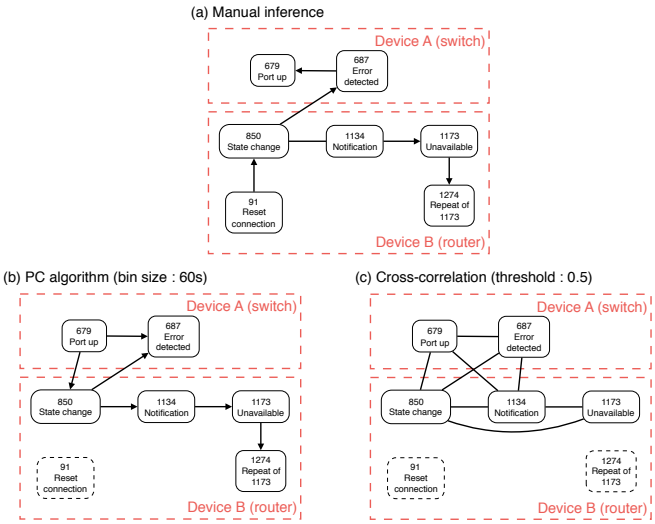

Fig. 10. Ground truth and detected causalities (BGP)

connected case shown in (d). This result indicates that the parameter tuning of the cross-correlation based method is not straightforward; a higher threshold misses important edges, and a lower one detect a large number of edges by chance.

2) BGP state initialization: Here, we introduce a more complicated failure spanning to two devices; An interface error on a router yields repeated BGP peering connections.

Figure 9 indicates the detected log templates in this failure. There are two log templates for one router and five templates for the other. Template IDs 679 and 687 show the network interface error, and appear in turns. ID 850 is a report of BGP state change at the counterpart caused by the network error. ID 1134 is a process of resetting BGP connection, which repeats with some unknown causes. ID 1173 shows BGP connection resetting failure, which appears sporadically on BGP connection resetting process. IDs 1274 and 91 are deriving events with BGP connection resetting, whose appearance is largely different from others. The root cause of BGP connection resetting does not appear in the system logs.

Figure 10 shows a plausible ground truth of this failure (a) and generated causality graphs with PC algorithm (b) and cross-correlation based method (c). We again confirm that the result of our method is closer to the ground truth than the best

TABLE V
FAILURE EVENTS REPORTED IN TROUBLE TICKETS AND THE NUMBER OF DETECTED EDGES IN OUR ALGORITHM

Event	Description	#edges
T1	Disconnected device	3
T2	Disconnected device	2
T3	FPC failure	2
T4	PIO error log	0

case of the cross-correlation based method even in the event spread between two devices.

Through the two case studies, our proposed method naturally extracted the important causal relations better than the traditional cross-correlation based method. One may think that graphs detected by the correlation method are still reasonable as shown in the figures, however, finding appropriate graphs by the correlation method is a needle in a haystack due to the sensitivity of the parameter tuning, in reality.

E. Comparison with trouble tickets

Finally, we compare detected events by the proposed algorithm to our trouble tickets data consisting of spatio-temporal information on failure events. The granularity of the events in the tickets is much larger than that of the syslog data. The ticket data contains 15 failure events for a month. The number of the syslog messages is $10,348,224$ in this period, and the number of detected edges is 554 before the post-processing and 61 after the post-processing. We do not find any raw log messages regarding 11 failures out of 15 failures.

Table V lists brief descriptions of the four failures in the trouble tickets and the number of edges detected by our method. We have managed to detect related log information for three failures; a disconnected network device from the backbone (T1 and T2), and a hardware failure of FPC (Flexible PIC concentrator) module in a router (T3). For the first case (T1 and T2), detected events belong to BGP (BGP state change and network interface updown), similar to the previous case study. For the second case (T3), the failure yields a causality in SNMP trap and FRU (Field-Replaceable Unit) state change. However, we fail to detect information about T4 (a PIC error). Log messages related to T4 appear only once without following events, which makes it impossible to investigate relations based on co-occurrence.

In summary, our proposed method reasonably managed to detect large events recorded in the trouble tickets if the related log messages are successfully recorded. However, it failed for finding an isolated event; this is also due to the lack of enough information for the inference, in principle.

VI. DISCUSSION

We demonstrate the effectiveness of our proposed approach to detect the causality of failure events. Compared to the cross-correlation method, PC algorithm extracts a smaller number of meaningful logs, thanks to conditional independence. However, as shown in § V-D, we observe a case when the edge direction is different from our intuition. For obtaining more appropriate results, we would additionally use timestamp information in the original data. However, we should carefully treat two events from different devices.

Comparing the detected results to the trouble tickets, we confirm that both results have a small overlap. However, our method missed some of the large events in trouble tickets because of the lack of related messages. Besides, the proposed algorithm detected small or medium size of failures while the trouble tickets mainly include serious failures directly affecting users. Thus, our method is useful in improving the daily network operation. In fact, it is not easy for network operators to identify the cause of login failures shown in the case studies.

Pre-processing before applying PC algorithm is an important step for getting appropriate results. 96% of the original messages are not adequate for the analysis because of their periodicity. Similarly, we divided original messages into temporal-spatial subsets to avoid detecting false causal relations. In fact, we confirm that there are no network-wide failures over a long time even in a large disaster case [29]. Similarly, recent literature points out that failures are isolated in time and space for many cases in a world scale network [1].

Post-processing is also a crucial step to pinpoint the target events, because PC algorithm detects causality but do not provide any importance on network operation. In our study, we rely on the simple heuristics to remove frequently appearing edges. However, as causal inference and anomaly detection are orthogonal, we plan to introduce or combine the current heuristics with a more sophisticated method to highlight the important events.

Currently, PC algorithm uses a binary time series as input due to a constraint of G-square test. As a possible improvement of the detection algorithm, we can additionally use information on the number of messages in a bin as introducing weights of the messages [30] (e.g., TF-IDF).

VII. CONCLUSION

In this paper, we propose a method to mine causality of network events from network log messages. The key idea of the work is leveraged on PC algorithm that reconstructs causal structures from a set of time series of events. Compared with a cross-correlation based approach, we confirm that PC algorithm outputs more appropriate events by removing pseudo correlations by chance with conditional independence. Furthermore, our pre- and post-processing steps are helpful in providing a small set of important events with causality to network operators. Through two case studies and comparison with trouble ticket data, we demonstrate the effectiveness of the proposed algorithm with 15 months syslog data obtained at a Japanese research and education network.

ACKNOWLEDGEMENT

We thank the SINET operation team for providing us the syslog and trouble ticket data. We thank Johan Mazel, Romain Fontugne, and Keiichi Shima for comments on this paper. This work is partially sponsored by NTT.

REFERENCES

[1] R. Govindan, I. Minei, ahesh Kallahalla, B. Koley, and A. Vahdat, "Evolve of die: High-availability design principles drawn from google's netwwork infrastructure," in *ACM SIGCOMM*, 2016, pp. 58–72.

[2] R. Gerhards, "The Syslog Protocol," RFC 5244, 2009.

[3] P. Spirtes and C. Glymour, "An algorithm for fast recovery of sparse causal graphs," *Social science computer review*, vol. 9, pp. 62–72, 1991.

[4] M. Kalisch and P. Bhlmann, "Estimating high-dimensional directed acyclic graphs with the pc-algorithm," in *The Journal of Machine Learning Research*, vol. 8, 2007, pp. 613–636.

[5] S. Urushidani, M. Aoki, K. Fukuda, S. Abe, M. Nakamura, M. Koibuchi, Y. Ji, and S. Yamada, "Highly available network design and resource management of sinet4," *Telecommunication Systems*, vol. 56, pp. 33–47, 2014.

[6] F. Salfner and M. Malek, "Using hidden semi-Markov models for effective online failure prediction," in *Proceedings of the IEEE Symposium on Reliable Distributed Systems*, 2007, pp. 161–174.

[7] K. Yamanishi and Y. Maruyama, "Dynamic syslog mining for network failure monitoring," in *ACM KDD*, 2005, p. 499.

[8] I. Beschastnikh, Y. Brun, M. D. Ernst, and A. Krishnamurthy, "Inferring models of concurrent systems from logs of their behavior with CSight," in *Proceedings of the 36th International Conference on Software Engineering - ICSE 2014*, 2014, pp. 468–479.

[9] Q. Fu, J. Lou, Q. Lin, R. Ding, Z. Dongmei, and T. Xie, "Contextual analysis of program logs for understanding system behaviors," in *Proceedings of the 10th Working Conference on Mining Software Repositories.*, 2013, pp. 397–400.

[10] T. Kimura, K. Ishibashi, T. Mori, H. Sawada, T. Toyono, K. Nishimatsu, A. Watanabe, A. Shimoda, and K. Shiomoto, "Spatio-temporal factorization of log data for understanding network events," in *IEEE INFOCOM*, 2014, pp. 610–618.

[11] R. Sipos, D. Fradkin, F. Moerchen, and Z. Wang, "Log-based predictive maintenance," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '14*, 2014, pp. 1867–1876.

[12] I. Fronza, A. Sillitti, G. Succi, M. Terho, and J. Vlasenko, "Failure prediction based on log files using Random Indexing and Support Vector Machines," *Journal of Systems and Software*, vol. 86, no. 1, pp. 2–11, 2013.

[13] Z. Zheng, L. Yu, Z. Lan, and T. Jones, "3-Dimensional root cause diagnosis via co-analysis," in *Proceedings of the 9th international conference on Autonomic computing - ICAC '12*, 2012, p. 181.

[14] K. Nagaraj, C. Killian, and J. Neville, "Structured Comparative Analysis of Systems Logs to Diagnose Performance Problems," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation - NSDI'12*, 2012, pp. 1–14.

[15] D. Heckerman, D. M. Chickering, C. Meek, R. Rounthwaite, and C. Kadie, "Dependency networks for inference, collaborative filtering, and data visualization," *Journal of Machine Learning Research*, vol. 1, pp. 49–75, 2000.

[16] A. A. Mahimkar, Z. Ge, A. Shaikh, J. Wang, J. Yates, Y. Zhang, and Q. Zhao, "Towards automated performance diagnosis in a large iptv network," in *ACM SIGCOMM*, 2009, pp. 231–242.

[17] B. C. Tak, S. Tao, L. Yang, C. Zhu, and Y. Ruan, "LOGAN: Problem Diagnosis in the Cloud Using Log-Based Reference Models," in *IEEE International Conference on Cloud Engineering (IC2E)*, 2016, pp. 62–67.

[18] J.-G. Lou, Q. Fu, Y. Wang, and J. Li, "Mining dependency in distributed systems through unstructured logs analysis," in *ACM SIGOPS Operating Systems Review*, vol. 44, 2010, p. 91.

[19] D. Yuan, H. Mai, W. Xiong, L. Tan, Y. Zhou, and S. Pasupathy, "SherLog : Error Diagnosis by Connecting Clues from Run-time Logs," *SIGARCH Comput. Archit. News*, vol. 38, pp. 143–154, 2010.

[20] C. Scott, S. Whitlock, H. Acharya, K. Zarifis, S. Shenker, A. Wundsam, B. Raghavan, A. Panda, A. Or, J. Lai, E. Huang, Z. Liu, and A. El-Hassany, "Troubleshooting blackbox SDN control software with minimal causal sequences," in *Proceedings of the 2014 ACM conference on SIGCOMM - SIGCOMM '14*, 2014, pp. 395–406.

[21] C. Scott, A. Panda, A. Krishnamurthy, V. Brajkovic, G. Necula, and S. Shenker, "Minimizing Faulty Executions of Distributed Systems," in *Preceedings of 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2016, pp. 291–309.

[22] P. Chen, Y. Qi, P. Zheng, and D. Hou, "Causeinfer: Automatic and distributed performance diagnosis with hierarchical causality graph in large distributed systems," in *IEEE INFOCOM*, 2014, pp. 1887–1895.

[23] T. Verma and P. Judea, "An algorithm for deciding if a set of observed independencies has a causal explanation," in *Eighth Conference on Uncertainty in Artificial Intelligence*, 1992, pp. 323–330.

[24] R. Vaarandi, "A data clustering algorithm for mining patterns from event logs," in *IEEE IPOM*, 2003, pp. 119–126.

[25] A. Makanju, A. N. Zincir-Heywood, and E. E. Milios, "Clustering event logs using iterative partitioning," in *ACM KDD*, 2009, pp. 1255–1264.

[26] M. Mizutani, "Incremental mining of system log format," in *IEEE International Conference on Services Computing*, 2013, pp. 595–602.

[27] S. Kobayashi, K. Fukuda, and H. Esaki, "Towards an NLP-based log template generation algorithm for system log analysis," in *Proceedings of The Ninth International Conference on Future Internet Technologies - CFI '14*, New York, New York, USA, 2014, pp. 1–4.

[28] R. E. Neapolitan, *Learning Bayesian Networks*. Prentice Hall Upper Saddle River, 2004.

[29] K. Fukuda, M. Aoki, S. Abe, Y. Ji, M. Koibuchi, M. Nakamura, S. Yamada, and S. Urushidani, "Impact of Tohoku Earthquake on R&E Network in Japan," in *ACM CoNEXT Special Workshop on the Internet and Disasters (WoID)*, 2011, p. 6.

[30] K. Fukuda, "On the use of weighted syslog time series for anomaly detection," in *IEEE IM*, 2011, pp. 393–398.