Mining causality of network events in log data

Satoru Kobayashi, Kazuki Otomo, Kensuke Fukuda, and Hiroshi Esaki

***Abstract*—Network log messages (e.g., syslog) are expected to be valuable and useful information to detect unexpected or anomalous behavior in large scale networks. However, because of the huge amount of system log data collected in daily operation, it is not easy to extract pinpoint system failures or to identify their causes. In this study, we propose a method for extracting the pinpoint failures and identifying their causes from network syslog data. The methodology proposed in this study relies on causal inference that reconstructs causality of network events from a set of time series of events. Causal inference can filter out accidentally correlated events, thus it outputs more plausible causal events than traditional cross-correlation based approaches can. We apply our method to 15 months' worth of network syslog data obtained from a nationwide academic network in Japan. The proposed method significantly reduces the number of pseudo correlated events compared with the traditional methods. Also, through three case studies and comparison with trouble ticket data, we demonstrate the effectiveness of the proposed method for practical network operation.**

***Index Terms*—Causal inference, Log data, Network management, PC algorithm, Root cause analysis.**

I. Introduction

Maintaining the stability and reliability of large-scale networks has been a fundamental requirement in network management. It is, however, not an easy task in practical networks because of the highly distributed, ever-evolving/growing, and heterogeneous nature of practical operational networks [2]. One effective way to track network status is to deploy monitoring agents in the network and collect log information corresponding to a change of system status. In operational networks, syslog [3] has been widely used for this purpose. Detailed log messages may provide a better understanding of system failures and their causes. Nonetheless, they are usually understood to be hard for network operators to identify because of a large amount of log data produced by a large set of network devices (e.g., routers, switches, and servers).

To solve this operational problem, various approaches have been developed to improve the performance of network monitoring and diagnosis using log messages. A simple approach is to cluster log messages related to a network event (e.g., failure) into a correlated group and analyze every group in detail. This approach assumes that the network event can yield a set of messages from some monitored network devices. One problem of log analysis is that co-occurrence of log messages does not always mean causal relations. The timestamp of messages is helpful in determining the causality, but the correctness of the timestamp need to be appropriately and accurately analyzed. Furthermore, network log messages appear more discretely and sparsely than other evaluation parameters (e.g., CPU or memory usage), which makes causality of events difficult to identify in network log analysis.

In this paper, we intend to extract causal relations beyond co-occurrences in log messages in order to identify important network events and their corresponding causes. For this, we first generate a set of time-series data from log messages on the basis of our preprocessing method. Then, we leverage a causal inference algorithm, called the PC algorithm [4], [5] named after P.Spirtes and C.Glymour. It outputs directed acyclic graphs (DAGs) that connect events with causality from a set of network logs. There are some challenges when applying causal inference algorithms to the network logs: (1) a large-scale network consists of multiple vendors' devices, and many types of messages appear in the set of logs: (2) messages occur discretely and sparsely, so they are not assumed in causal inference algorithms, and (3) not all detected causalities are necessarily important in the context of network management.

To overcome these technological challenges, we propose a mining algorithm built on the causal inference. We apply our proposed algorithm to 15 months' worth of long syslog messages, 34 million messages in total, collected from a nationwide research and education network in Japan [6]. We obtain a reasonable number of causal edges with lower false positive rates than a commonly used traditional cross-correlation based method. Furthermore, we design simple heuristics that suppress commonly appearing causality and highlight uncommon causal relationships. Through case studies and a comparison with trouble ticket data, we demonstrate the effectiveness of our proposed approach.

The contribution of this paper is twofold: (1) we propose a log-mining system based on causal inference for log data of highly complicated networks (§ III, § IV), i.e., a wide variety of log messages sparsely generated by heterogeneous network devices: and (2) after careful validations (§ VI), we show that our proposed method can report meaningful events, sufficiently suppressed into a reasonable number for operators to read, with causal relations from a huge number of log messages (§ VII), which is practically useful for daily network operation.

II. Related Work

Prior literature has been devoted to automated troubleshooting and diagnosis with system logs. Automated system log analysis has contributed to solve various problems (e.g., system failures [7], [8] and security threats [9], [10]). Some works have conducted contextual analysis of log data to retrieve useful information in troubleshooting for network management. Contextual log analysis can be classified into four groups: model, spatial, relational, and co-operative approaches.

The model approaches present system changes behind log events as some models, especially state transition models.

S. Kobayashi, K. Otomo and H. Esaki are with the Graduate school of Information Science and Technology, the University of Tokyo, Tokyo, JP
K. Fukuda is with National Institute of Informatics, and Sokendai.

[0]This paper is an extended version of work published in Ref. [1].

Salfner and Malek [11] use a hidden semi-Markov model for failure prediction. Yamanishi and Maruyama [12] analyze system logs with a multidimensional hidden Markov model. Beschastnikh et al. [13] generate finite state machines from execution traces of concurrent systems to provide insights into the system for developers. Fu et al. [14] 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. [15] characterize network faults in terms of log type, time, and network devices with a tensor analysis of system logs. Sipos et al. [16] use a multi-instance learning approach to extract system failures from system logs. Fronza et al. [17] predict system failures by classifying log events on the basis of 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 a graph approach, and are especially useful for fault diagnosis. Root cause analysis with system logs has also been a hot topic in the context of network management. A common approach is to infer causal relations among events in system logs. Zheng et al. [18] detect correlated events in system logs in a supercomputer system and remove pseudo correlations with conditional independence. Nagaraj et al. [19] generate dependency networks [20] (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 use a probabilistic method to find conditional independence. Mahimkar et al. [21] take an approach to extract failure causality from correlations. They use multiple regression coefficients, but this approach requires large processing time for a large number of events. Some approaches estimate causal relations without causal inference.

Some studies use different approaches to detect root causes of trouble in system logs. Tak et al. [22] 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. [23] 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. [24] pinpoint errors in source code by matching a function call graph with error log messages. Scott et al. [25], [26] troubleshoot software defined network control software on the basis of causal relations estimated with external causal inference tools. These approaches need the external data to be available on the systems.

Our work in this paper is categorized into the relational approaches based on causal inference. However, existing work has used much resource to estimate causal relations. We propose an efficient causal inference algorithm based on graph approaches on system logs.

Closer to our work, Zheng et al. [18] leverage causal inference on pinpointing root causes of network events. The main difference from ours is how to apply causal inference; they search conditional independence heuristically to decrease processing time. In that way, one can find only a part of causality among log events that is enough for determining root causes. In contrast, our approach investigate all the causal relations because edges not indicating root causes will also help system operators understand the system behavior as shown in our case studies.

In this paper, we use the PC algorithm [4], [5], which is a method to estimate directed acyclic graphs (DAGs) from statistical data on the basis of conditional independence. In the field of network measurement, Chen et al. [27] also use this algorithm. They generate causality graphs of network events from some monitored parameters such as RTT and TCP window size. System log messages occur more discretely and sparsely than these parameters. This difference requires elaborate preprocessing methods.

III. Causal inference

The key idea of our proposal is to detect causality of two given events in the network logs. The causality is a partial order relationship different from correlation, which is typically quantified by a correlation coefficient. Using correlation as causality yields many false positives because a positive correlation between two events does not always mean causality. Checking timestamps of two correlated events is helpful in determining causality, but the timestamp of system logs is not always reliable for determining causal directions due to NTP's time synchronization error, jitter, and network failures. Thus, we need to estimate causal directions among events without timestamps, in theory. Such algorithms have been developed in the field of causal inference. In this section, we introduce PC algorithm [4], [5], a generalized causal inference algorithm, to infer causal relationships from a set of events.

We first introduce the concept of conditional independence. Conditional independence is a key idea to reveal causality from correlation. 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 is eliminated by considering the related event C. Note that C can represent multiple events. If A and B are conditionally independent for given C, we can consider that A and B have no causality, because their relation is indirect across C. With the idea of conditional independence, we can remove pseudo causality (i.e., false positives) in this way.

The PC algorithm [4], [5] is a graph-based method to reconstruct causal relationships among nodes with conditional independence. For efficient processing, it assumes that nodes form no loops of edges in the estimated graph. Here, its output causality graph is a DAG of events corresponding to the causality of events. The PC algorithm consists of four steps (see also Figure 1).

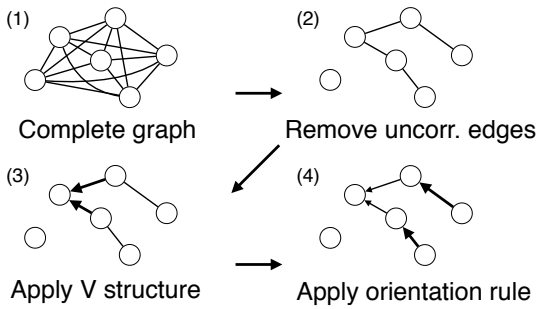


Fig. 1. Causal inference in the PC algorithm

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 on the basis of V-structure.
4) Determine edge direction with orientation rule.

In the first two steps, we estimate a skeleton graph that is an undirected causality graph with the idea of causal inference. For testing conditional independence in step 2, we first cut edges (say $A - B$) the nodes of which are conditionally independent without another additional node, i.e., this is a special case of Equation 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 Equation 1. We remove the edge if at least one of the other nodes holds the conditional independence. Repeating this process, we find out which edges indicate some causal relations among all nodes.

In the latter two steps, we determine directions of the detected edges with two rules. In step 3, a V-structure is a condition to decide on the direction of an 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, the orientation rule prevents a loop being made among events by the definition of DAG [28]. Some edges can be undirected even after applying the PC algorithm if one does not have enough information to determine edge directions.

In practice, the PC algorithm has some variations. The original-PC [4] has a problem that output depends on the order of input data. In contrast, stable-PC [29] is order-independent although it takes longer processing time. Parallel-PC [30] requires shorter processing time with parallel processing. In addition, the PC algorithm has been extended to improve its accuracy [31]. In this paper, we use the stable-PC algorithm to generate DAGs from system logs.

IV. Methodology

To detect root causes of network events in system logs, we intend to infer causal relations among log events obtained at an operational network. Here, we first provide an overview of the proposed system and then describe in detail preprocessing, calculation of conditional independence, and postprocessing.

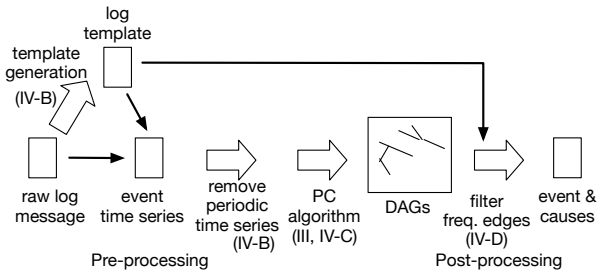


Fig. 2. Processing flow

```
- oritinal message
sshd[1234]: Accepted password for ops from 192.0.2.3
    port 12345 ssh2
- log template
sshd[**]: Accepted password for ** from ** port ** **
```

Fig. 3. Example of log template

A. System overview

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

First, we generate a set of input time-series of the PC algorithm with some preprocessing techniques: log template generation to translate messages into time-series, and periodic event filtering to reduce detected relations of invaluable events for troubleshooting (§ IV-B). Next, we apply the stable-PC algorithm [29] to the time-series. In this step, we discuss two conditional independence testing methods in the PC algorithm (§ IV-C). As a result, we obtain DAGs with nodes of event time-series and edges of causal relations. For practical use, we use a postprocessing technique after the PC algorithm to focus on abnormal behaviors in the system (§ IV-D).

B. Preprocessing

To apply PC algorithm to log data, there are two issues to be carefully considered.

First, raw log messages cannot be analyzed directly with statistical approaches because they are string, not numeric. Log messages consist of timestamps, source hostnames, and partially formatted messages. For input of the PC algorithm, we generate a set of time series corresponding to each event from the raw log messages obtained from one network device.

Second, the distribution of log appearances is long-tailed; some events, like daily processes, appear much more frequently than others, like error logs. This difference in frequency greatly affects the PC algorithm. Indeed, the PC algorithm detects more edges from frequent events while hiding important relations of minor events. To avoid this, we remove events that appear frequently but are less important in troubleshooting.

To overcome the two problems, we apply the following preprocessing to a set of log messages. First, we extract log templates from the original log messages except timestamps and source hostnames. The log template is a log format the variables of which (e.g., IP address, port) are replaced by a wild card (*); it is a standard (i.e., comparative) format of log

messages. Figure 3 is an example of a log message and the corresponding log template. In this example, variables such as process ID, user name, IP address, port number and protocol name are replaced by wild cards. There have been many log template generation algorithms for log messages [32]–[35]. We use a supervised learning-based log template generation algorithm [35]. This algorithm requires manually annotated training data but generates more accurate log templates than other clustering-based algorithms because they tend to fail to classify minor logs, which are still important in causation mining for troubleshooting. The supervised algorithm [35] largely solves this problem. We use a set of templates that are manually checked after applying the algorithm. We finally extracted 1,789 templates from 35M log messages in our dataset (see also § V). More examples of generated log templates are also shown in Figure 13, Figure 15 and Figure 17. Next, we construct a set of event time series generated by each log template per device (router or switch) from all log data. In other words, each time series contains the number of appearances of one log template from one device in a disjointed (non-overlapping) time bin (size b, ghich we use for 60s). The generated time series depends on the conditional independence test method (see § IV-C). We discuss the parameter dependency of bin size b and its overlaps for the edge detection in § VI-C.

We then remove a large number of unrelated event time series indicating strong temporal periodicity. Such a periodic log template represents regular daily events caused by an event timer (e.g., cron) and would be a source of misdetection of the causality. However, a periodic event often contains outliers that do not follow its whole periodicity. As these outliers are also expected to be important for troubleshooting, we leave them in time series after removing periodic events. For this purpose, we use following two methods: a Fourier analysis and a linear regression analysis.

First, we find periodic events with the Fourier analysis. If a given time-series $f(t)$ of N length with a time bin (size b_f) is periodic, its power spectrum is characterized by equally spaced peaks. The power spectrum A of the time-series is defined as $A_k = |\sum_{n=0}^{N-1} f(n)e^{-2i\pi\frac{nk}{N}}| \quad (k = 0, 1, \cdots, N-1)$. We pick up the first ℓ peaks of the power as the candidates of the following investigation (empirically, we set $\ell = 100$). Let the intervals of each peak P, $f(t)$ be considered as periodic if $\frac{\sigma(P)}{\bar{P}} < th_p$ where $\sigma(P)$ is the standard deviation of P, and $\bar{P}$ is its average (we use 0.1 as th_p). This condition intuitively indicates that the intervals of all peaks are equally distributed if $\sigma(P)$ is sufficiently smaller than its average. Thus, if the peaks P of A_k meet this condition, the given time-series $f(t)$ is considered as periodic.

Then, we construct residuals in periodic events with inverse Fourier transform. As a periodic component of $f(t)$ is presented in the peaks of power spectrum A_k, the residual component A' is presented as A_k for $A_k \leqq th_a max(A)$, and 0 otherwise. This component cuts off peaks (i.e., periodic component) by threshold th_a (we use 0.4 as th_a). Then we obtain the inverse Fourier transform $g(t)$ of A'. Here, the residual time series $h(t)$ is presented as $f(t)$ for $f(t) > \frac{g(t)}{2}$, and 0 otherwise.

Next, we find missing periodic events with a linear regression analysis. In our preliminary analysis, we found that the Fourier analysis misses periodic events of short intervals or non-fixed intervals with noise. The cumulative time series of these quasi periodic events is an approximately linear increase. Thus, we compare the time series $f(t)$ to its linear regression $l(t) = \frac{tm}{N}$ where $m = \sum_{t=0}^{N-1} f(t)$. The time series is considered as periodic events if $\sum_{t=0}^{N-1} \frac{(f(t)-l(t))^2}{mN} < th_l$. The left side is an index of the difference between the time series and its linear regression, and we compare the index with the threshold th_l to judge whether the difference is small enough (we use 0.5 as th_l). With this method, we find out which time series sufficiently fits its linear regression. This method cannot find periodic events of large intervals but works well with Fourier analysis because these two methods mutually compensate for their disadvantages.

C. Calculation of conditional independence

Here, we discuss how to calculate the conditional independence for the PC algorithm. As described in § III, the PC algorithm tests conditional independence of nodes X and Y with another node Z. One can consider two methods for testing conditional independence: G-square test (discrete) [36] and Fisher-Z test (continuous) [36]. Other methods (like using probabilistic causality theory [18]) are not reasonable to use in the PC algorithm because they usually require extra information for processing (like time-series order of events).

1) G-square test: The G-square test is a method to evaluate conditional independence of binary (consisting of zero and one) or multi-level data. This method is a natural extension of a chi-square test and is based on information theory, using cross entropy. The G-square statistic G^2 is defined as:

$$G^2 = 2mCE(X, Y \mid Z), \tag{2}$$

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 (i.e., $x(t)$, $y(t)$, and $z(t)$). We check a p-value of the null hypothesis of the test with a threshold $p = 0.01$. For the G-square test, we use a binary event time series generated from the original event time series by converting each value for $x(t) \geqq 1$ into $x(t) = 1$.

The conditional cross entropy is calculated from the count of each value in the input data. Assume that the probability is $P(i, j \mid k)$ and the count of t is S_{ijk} where $x(t) = i$, $y(t) = j$, and $z(t) = k$. Here, the G-square statistic G^2 can be transformed as follows:

$$G^2 = 2m \sum_{ijk} P(i, j \mid k) log \frac{P(i, j \mid k)}{P(i \mid k)P(j \mid k)} = \sum_{ijk} 2S_{ijk} log \frac{S_k S_{ijk}}{S_{ik} S_{jk}}. \tag{3}$$

This property enables the G-square test to calculate faster, but also limits the format of input data to binary.

2) Fisher-Z test: The Fisher-Z test evaluates conditional independence on the basis of Pearson's correlation coefficient. This test is a combination of two statistical techniques: Fisher-Z transformation to estimate a population correlation coeffi-

cient, and a partial correlation to evaluate the effect of other nodes. The statistic Zs is defined as:

$$Zs = \frac{\sqrt{m - |Z| - 3}}{2} log \frac{1 + r}{1 - r}, \quad (4)$$

where m is the size of time series, r is partial correlation of X and Y given Z, and $|Z|$ is the number of nodes to consider with X and Y. Zs statistic corresponds to a standard normal distribution for $r = 0$. We check the p-value of the null hypothesis of the test with a threshold $p = 0.01$.

Compared with the G-square test, Fisher-Z has an advantage in terms no constraint for the input data. For log analysis, we consider the number of events in each bin.

Recall that, by the nature of the two conditional independence methods (binary or continuous), the input time series for the PC algorithm must be different for the G-square and Fisher-Z tests as shown in Figure 4.

D. Postprocessing

The output of the PC algorithm is a set of causalities. However, the PC algorithm does not provide any information on the importance of events. Thus, we require a further step to filter commonly detected (or uninteresting) causality from the provided information for operators.

In this study, we simply count the number of appearances of frequently appearing edges, because of the long-tailed nature of log message appearance; i.e., the same edges appear in many devices over time. Thus, we remove frequently appearing edges with a threshold for easily identifying unusually important causality. Note that this postprocessing is supposed to affect only the superficial notifications for operators. We discuss the effect of the postprocessing in § VII-B.

V. Dataset

To evaluate the effectiveness of our approach in the previous section, we use a set of backbone network logs obtained from a Japanese research and educational network (i.e., SINET4 [6]) that connects over 800 academic organizations in Japan. This nationwide network consists of 8 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 devices in SINET4, though some 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 a syslog protocol. We analyze 456 day-long consecutive logs composed of 35M log messages from 2012-2013.

To easily understand log messages for pinpointing events and their 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 six groups: System, Network, Interface, Service, Management, and Monitor. System shows internal processes of devices, like operating system and hardware modules. Network is linked to some protocols for communication. Interface provides the status of physical or virtual network interfaces. Service consists of network services provided to other devices, like NTP. Management corresponds to configuration changes by operators. Monitor contains functions that monitor system behaviors, like SNMP and syslog. Also, we separately introduce some external groups related to frequently used protocols and services.

TABLE I
Classification of log messages

Type	#messages	#preprocessed	#templates
System	28,690,829	1,561,460 (5%)	550
Network	1,259,237	241,923 (19%)	166
Interface	238,848	231,918 (97%)	191
Service	213,853	23,616 (11%)	35
Mgmt	4,114,148	118,124 (29%)	590
Monitor	157,979	71,154 (45%)	87
VPN	21,979	21,391 (97%)	89
Rt-EGP	21,539	21,027 (97%)	66
Rt-IGP	4,166	3,995 (96%)	15
Total	34,722,578	2,294,608 (7%)	1,789

In the table, the column "#messages" represents the number of raw log messages, "#preprocessed" is the number of processed messages (after the preprocessing), and "#templates" is the number of identified log templates. We see major log messages are System events. These groups are related to repeatedly reported events like cron. Management also has a large number of log templates because this group consists of configuration change events that record various settings in the messages. Also, the numbers of log templates of VPN and Routing are larger than the event appearances of the groups. It is reasonable that these protocols output messages only when the configurations or states are changed. The template generation algorithm outputs 1,789 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 (§ VII-D).

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 connected to the core router. We analyze every one-day-long log data in the dataset because we target short-term causality instead of long-term causality. We will discuss this window size dependency on § VI-D. Finally, we generate 3,648 DAGs (456 days and 8 network subsets) from the entire dataset.

VI. Validation

In the previous section, we described a series of techniques to extract causal relations from system logs. We still have some challenges to consider in practical use through these techniques, like conditional independence test methods and parameters. Here, we investigate the appropriateness of preprocessing (§ VI-A), the difference between conditional independence tests (§ VI-B), and the parameter dependencies (§ VI-C and § VI-D). In addition, we validate the possible false positives of the PC algorithm (§ VI-E).

For the experiments, we use a computer with an Ubuntu 16.04 server (x86_64) equipped with Intel(R) Xeon(R) X5675 (3.07GHz) and 48GB memory. The implementation of our

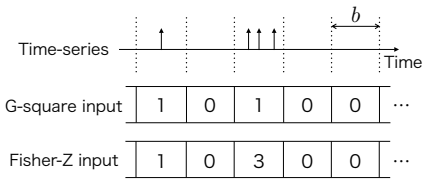


Fig. 4. Input time series for two conditional independence tests

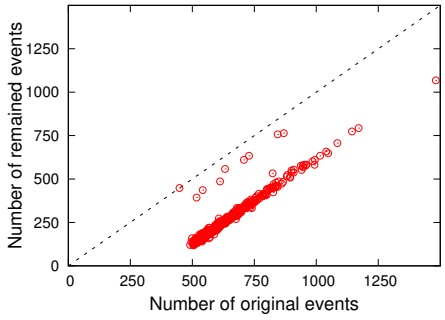


Fig. 5. Preprocessing: event reduction

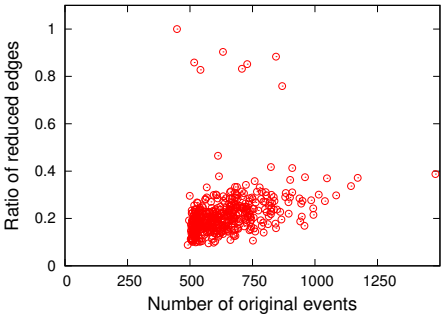


Fig. 6. Preprocessing: edge reduction

proposed method can be downloaded from https://github.com/cpflat/LogCausalAnalysis.

A. Preprocessing

As mentioned in § IV-B, we remove periodic events from the input time-series of PC algorithm in order to reduce relations of less important events for troubleshooting. Our preprocessing methods took 173 minutes for processing all datasets for the 456 days. Here, we investigate the filtered events and their impacts on the resulting DAGs.

First, we show how our preprocessing helps to reduce of the number of messages. Figure 5 shows the scatter plot of the number of events suppressed by the preprocessing and the number of the original events. Every dot represents an one-day data. The diagonal is a baseline to indicate no preprocessing.

Our preprocessing reduces the constant number of events which is independent from the number of input events. Thus, the method removes regular events that appear every day. This is not contrary to our intuition because periodic events are usually regular as long as there are no system changes.

Table I shows the breakdown of the preprocessed messages. We find that System is mostly removed by preprocessing since this group includes cron events that work periodically. Service is also omitted because of frequently appearing NTP synchronization messages. Similarly, Network, Management and Monitor are largely suppressed. In total, 7% of log messages are used as input of the PC algorithm. These removed events, like cron and NTP, are reported much more frequently than other events and contribute to system troubleshooting in fewer cases. On the other hand, the events of Interface, VPN, and Routing are seldomly removed. Their messages represent the changes of states or configurations, which do not have periodicity and are important information for troubleshooting. Thus, our preprocessing techniques definitely remove periodic and unnecessary events while leaving important events in the input time-series of the PC algorithm.

Similar to the number of events, we focus on the number of edges reduced by the preprocessing. Figure 6 shows the reduction rate of edges with the preprocessing method. We can see that the preprocessing method removes 70% to 90% of edges from generated DAGs. Generally, periodic events likely form more edges than other events because periodic events with the same interval have a similar event appearance

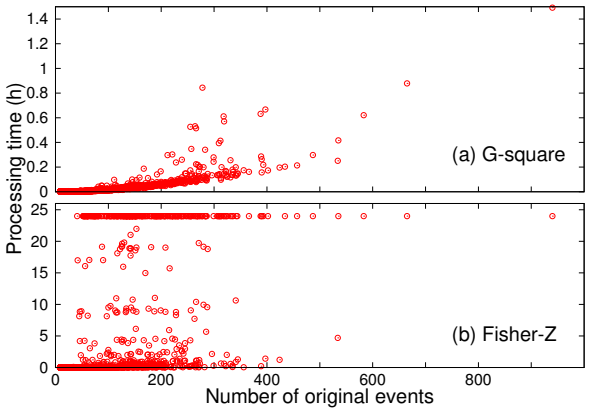


Fig. 7. Processing time of the PC algorithm

in their time-series. In addition, such periodic events usually appeared regularly. Thus, the number of detected edges among the periodic events is large. However, these relations are not usually important in troubleshooting due to their constancy. If the majority of provided information is given to operators, it will compromise the practical usefulness. This is why we need preprocessing before the PC algorithm.

B. Difference in conditional independence

In § IV-C, we introduced two reasonable methods to evaluate conditional independence: the G-square and Fisher-Z tests. These two methods are different in terms of statistical basis and input data format, which greatly affect the results of the PC algorithm. We compare the processing time and the results of these two conditional independence test methods.

First, we investigate the processing time of the PC algorithm with the two methods. Figure 7 shows the distribution of processing time of the PC algorithm. In this experiment, some points are marked as a day (24h) when the processing did not finish within one day in our environment due to a large number of events. In Fisher-Z, 7% of the datasets (253 out of 3,648) timed out. Note that timeout does not always occur in a large dataset. This means the processing time of Fisher-Z depends largely on the complexity of DAGs (i.e., the number of edges) rather than the number of events. In contrast, the PC algorithm processes the event dataset in reasonable time

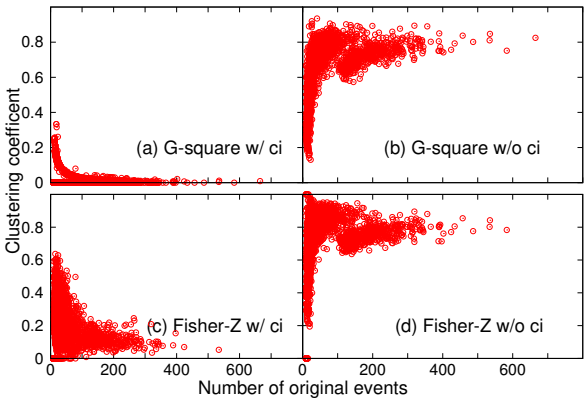


Fig. 8. Clustering coefficient of DAGs generated with the PC algorithm

with the G-square test though the number of events per day increases faster than linearly.

Next, we analyze the generated DAGs with the two conditional independence methods. Table II shows the number of detected edges. The Fisher-Z test detected many more edges than the G-square test. This result suggests that the Fisher-Z test is a better method for the PC algorithm than the G-square test in terms of detectability. However, in our manual investigation, edges that are detected with Fisher-Z but not with G-square are not useful for troubleshooting because most such edges provide redundant information. In addition, we obtain more than 100 times edges without considering conditional independence (i.e., only correlation), and it is easily expected that most of these edges are false positives. Here, we need to examine a quality of generated DAGs (i.e., the number of redundant or false positive edges) instead of the number of edges. To investigate the quality of DAGs, we show additional results with two metrics: global cluster coefficient and max clique size.

First, we analyze the quality of generated DAGs with a global clustering coefficient [37]. A global clustering coefficient is defined as the ratio of triangles to the possible number of them in a given graph. A large coefficient means that the conditional independence test cannot cut edges effectively. Figure 8 shows the distribution of a global clustering coefficient of generated DAGs. This graph compares four methods: Fisher-Z and G-square with and without considering conditional independence (only see relations of every two nodes, corresponding to analyzing not causality but correlation). The goal of our work is to reveal causality of log events with small false positives in practice, so the cluster coefficient is expected to be small enough. Fisher-Z and G-square without conditional independence generate DAGs with a high cluster coefficient (close to 1.0). These DAGs form more complete subgraphs of events that appear closely in time-series. This means that conditional independence must be considered to remove false positives in the causality graphs and shows a crucial limitation of the traditional correlation algorithm. We can also see that Fisher-Z generates DAGs at least twice as dense as those of G-square.

Second, we compare the max size of clique (i.e., complete subgraph) in DAGs, in order to check the quality of DAGs. Figure 9 shows the distribution of max clique size of generated DAGs without considering edge directions. This figure also shows that G-square generates fewer and smaller complete subgraphs than Fisher-Z. At best, G square forms triangular subgraphs in DAGs. In contrast, Fisher-Z forms five or more complete subgraphs, which does not seem useful as causality.

These results indicate that Fisher-Z forms more triangle edges than G-square. However, the information from such triangle edges actually is redundant in troubleshooting. There are no reasonable situations where more than three events equally depend on each other. At least, one triangle edge is an indirect relation that is a false positive or non-meaningful information for system operators. In Figure 8 and Figure 9, edges that are detected with Fisher-Z but not with G-square form such edge triangles. Thus, the distinctive edges in Fisher-Z are not important for troubleshooting in many cases (see also § VII-C3, which is a good example that Fisher-Z forms redundant edges).

It is reasonable to assume Fisher-Z fails to remove false positive edges well. Fisher-Z requires its input data to be distributed normally [38]. If the data is large enough like access logs, Fisher-Z works appropriately. However, our log events appear sparser in time-series. In this case, Fisher-Z cannot reject the existence of causality with conditional independence, leading to leaving more edges in DAGs.

We remark on the processing time in both tests. The PC algorithm requires longer processing time to search for conditional independence if many edges are left in the skeleton estimation step. The PC algorithm conducts a recursive search of conditional independence while increasing the number of condition nodes. A complete graph is the worst case of this algorithm that requires $O(n^{\frac{n}{2}})$ of computation time (where n is the number of nodes). According to Figure 9, Fisher-Z generates some complete subgraphs that are large enough to lead to combinatorial explosions. This is the main reason for yielding timeout in Fisher-Z in Figure 7. Thus, we conclude that G-square is better than Fisher-Z at evaluating conditional independence if the input dataset is sparse in time-series.

C. Bin size dependency

In the previous section, we explained the basic flow of the processing. However, the performance of the algorithm depends on the parameter setting. The most important parameter in the algorithm is the time bin size b for aggregating events. Intuitively, larger b should detect more false positive edges in a causality graph because unrelated events can be co-located in the same bin. Similarly, smaller b should make it difficult to detect causality between two events with a certain delay. In addition, the PC algorithm with smaller b takes longer processing time because it depends proportionally on the total number of bins and the dataset size is fixed. Thus, we need to determine a reasonable size of b in practice. To investigate the dependency of the bin size b on the number of detected edges, we carefully observe the outputs of the PC algorithm for different bin sizes from 10s to 300s.

We first investigate the number of detected edges with non-overlapping bins of the five different bin sizes. Table III lists

TABLE II
DEPENDENCY OF THE CONDITIONAL INDEPENDENCE METHOD ON THE NUMBER OF DETECTED EDGES

Method	Directed edges	(Diff. device)	Undirected edges	(Diff. device)	All edges	Edges without CI
G-square	1,617 (10.0%)	508 (3.1%)	14,579 (90.0%)	1,630 (10.0%)	16,196	1,779,074
Fisher-Z	49,710 (61.7%)	20,221 (25.1%)	30,856 (38.3%)	9,646 (12.0%)	80,566	1,857,753

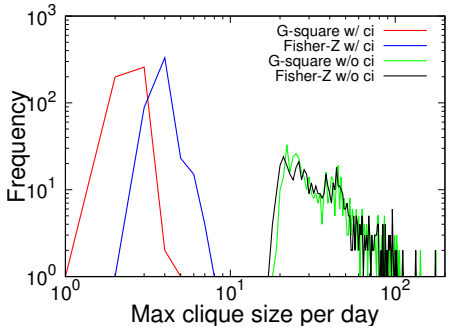


Fig. 9. Distribution of max clique size of DAGs generated with the PC algorithm

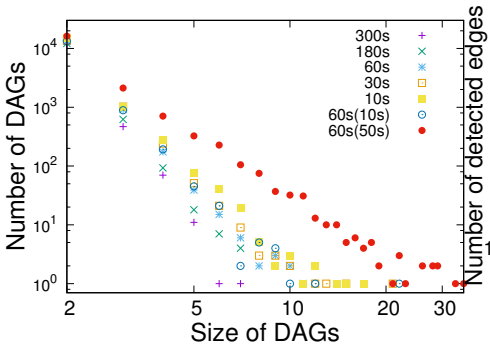


Fig. 10. Distribution of size of DAGs

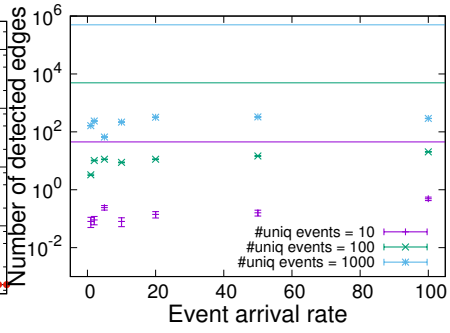


Fig. 11. The number of false positive edges in a Poisson arrival process

TABLE III
DEPENDENCY OF BIN SIZE AND OVERLAP ON THE NUMBER OF EDGES

Bin	Overlap	Directed edges	(Diff. device)	Undirected edges	(Diff. device)	All edges
300s	0	729 (5.5%)	261 (2.0%)	12,372 (94.4%)	1,060 (8.1%)	13,101
180s	0	878 (6.3%)	308 (2.2%)	13,012 (93.7%)	1,190 (8.6%)	13,890
60s	0	1,617 (10.0%)	508 (3.1%)	14,579 (90.0%)	1,630 (10.0%)	16,196
30s	0	2,036 (12.0%)	570 (3.3%)	14,979 (88.0%)	1,657 (9.7%)	17,015
10s	0	2,650 (14.1%)	614 (3.3%)	16,078 (85.9%)	1,358 (7.2%)	18,728
60s	10s	1,623 (10.0%)	486 (3.0%)	14,612 (90.0%)	1,495 (9.2%)	16,235
60s	50s	6,862 (25.5%)	3,121 (10.8%)	22,023 (76.2%)	4,243 (14.7%)	28,885

the total number of detected directed and undirected edges from the dataset. The column (Diff. device) represents the edges between two different devices. First, we can see that the number of detected edges is smaller for larger b. The result seems to differ from our intuition that larger b detects more false edges. As described in § IV-C, cross entropy affects the result of the G-square 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 the same log templates is not considered in our binary time series. Thus, the cross entropy becomes smaller for larger b, resulting in more rejections of the test. Note that the reduction of edges with larger b does not mean a higher accuracy of detected edges. Larger b reduces the edges between incompletely synchronized events, like the events of different functions. The edges between these events are likely to be useful in troubleshooting. Second, we find a decrease in the number of detected edges with smaller b. In particular, the decrease in the number of undirected edges between two different devices is clear. This result agrees with our intuition, because the edges between different devices usually have a larger delay than the others because of the network latency.

Next, we discuss overlapping bins. For non-overlapping bins in the above experiments, we may miss some useful edges if two causal events are located in neighboring bins. We investigate the dependency of DAGs on the length of bin overlap. Here, we fix the bin size b to 60s, which detects a sufficient number of edges with non-overlapping bins, and then change the overlap length. For instance, if we use 10s overlap, a new bin starts at 50s in the current bin. Note that the number of total bins is larger with overlapping bins than that with non-overlapping bins. Table III lists the number of detected edges with bin overlap size 10s and 50s. With 10s-bin overlap, we detect an almost similar number of edges to the non-overlapping bin, meaning that the boundary issue is negligible. In contrast, we find a greater number of edges with the 50s overlap. In this case, the total number of bins is equal to that with 10s-bin without overlap. Even compared with that, we see more edges with the 50s overlap. From the results, we find two issues to determine b: (1) smaller b to obtain a sufficient information gain, and (2) larger b to detect related events with a large time lag. Using bins with large overlaps solves this problem; large bin overlaps increase the total number of bins for more information gain, and the causal events with a certain time lag are fit in the bin.

Furthermore, we examine the size of DAGs produced by the PC algorithm. We focus on connected subgraphs, i.e., consisting of at least two nodes. Figure 10 shows the DAG size distribution found in all log data; as expected, most

TABLE IV
DETECTED EDGES OF PC ALGORITHM WITH CHANGING BIN SIZE AND DATA WINDOW SIZE

Window size	#Average edges	#Unique edges
24 days	266	2900
7 days	133	3149
3 days	83	3094
1 day	36	2936

DAGs are small and a few are large. We find that smaller b generates more connected subgraphs. This is consistent with the previous result that a larger number of edges is detected for smaller b. Similarly, we find more connected subgraphs using 60s-bin with 50s overlap than that using non-overlapping bins of any sizes. 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 an NTP server fails 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. This connected event appears frequently and its causality is false positive.

Over all, these results suggest that we should use a reasonable size of bin with large overlap. Thus, we use $b = 60s$ with 50s overlap for further analysis (shown in § VII).

D. Window size dependency

In the previous section, we investigated bin size of the input time-series of the PC algorithm. There is another parameter for the PC algorithm to consider: the window size of input time-series. As described in § V, we divide a dataset into small ones with a window size to emphasize temporal relations of events. Also, processing time will greatly increase without dataset division, because one DAG contains too many nodes (i.e., events). For these, we use one day long time-series as the input for the PC algorithm. We need to investigate the effects of window size on the results of the PC algorithm.

Table IV compares four window sizes in processing datasets of all 456 days. "Average edges" is the average number of detected edges in each input data of a given window size. "Unique edges" is the total number of different edges (i.e., edges between nodes of different log templates and different source devices) in all datasets. We can see that the number of unique edges is consistent in every window size. This result indicates that the window size does not largely affect the detectability of edges. Thus, it is still reasonable to use one-day-long time-series as an input of the PC algorithm.

E. False positives

We consider two types of major false positives in the results of the PC algorithm. One is spurious correlation edges: the edges between nodes that are correlated but have no causation. This type of false positive is caused by the failure of a conditional independence test. We already investigated this in § VI-B. The other is falsely connected edges between accidentally co-occurring events. The PC algorithm has difficulty distinguishing between truly causal events and accidentally co-occurring events. Here, we investigate a possibility that two events are accidentally connected by an edge in the PC algorithm. For synthetic multiple random events without causality, all detected edges are clear false positives by chance.

We prepare 10, 100, and 1,000 randomly generated unique events that occur in accordance with follows a Poisson process with different arrival rates. Applying our method to these surrogate time series (b = 60s), we count the number of edges detected by the PC algorithm. These edges are all considered as false positives, because all events are generated independently of each other. For 10 unique events (also 100 and 1,000), the maximum number of detected edges among nodes is 45 (4,950 and 499,500), i.e., the worst case.

Figure 11 illustrates the number of falsely detected edges for different event arrival rates. Each point is the average of 100 trials with unique random events. We can find that the number of false positives is small and fairly stable over different arrival rates. The false positive ratios are 1.1% for 10 unique events, 0.4% for 100 events, and 0.06% for 1,000 events. Thus, we may find at least about 1% of false edges among events following Poisson processes.

Events in real datasets may not follow Poisson processes in many cases. For example, there are events with partially periodic or linear time-series. These events do not clearly follow Poisson processes, and they form more false positive edges than random events in our investigation. Here, the preprocessing method to remove periodic or linear time-series is necessary to decrease false positive edges.

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

Type	All Edges	(Direct)	Edges of Inner-types	(Direct)	Important Edges	(Direct)
System	13,874	583	12,556 (91%)	380	1,206 (9%)	303
Network	1,081	142	500 (46%)	36	621 (57%)	103
Interface	2,137	349	1,602 (75%)	256	447 (21%)	130
Service	261	28	134 (51%)	16	179 (69%)	24
Mgmt	11,188	937	10,008 (89%)	732	1,299 (12%)	234
Monitor	344	57	194 (48%)	28	274 (80%)	49
VPN	1,126	90	1,084 (96%)	74	204 (18%)	52
Rt-EGP	2,361	1,047	2,170 (92%)	970	216 (9%)	206
Rt-IGP	20	1	16 (80%)	0	20 (100%)	1
Total	32,392	3,234	28,264 (87%)	2,492	4,808 (15%)	1,102

VII. RESULTS

In the previous section, we analyzed the parameter dependency of the algorithm on the number of detected causalities. This section describes the distribution of detected edges in the semantic classification (§ VII-A) and the effectiveness of our postprocessing (§ VII-B). Then we will present three case studies (§ VII-C) and a comparison with trouble tickets (§ VII-D) for further detailed understanding.

A. Detected edges

We first list the classification of detected edges (Table V) from the preprocessed log messages (Table I) by the PC

algorithm. Table V shows the number of neighboring nodes (i.e., events) for the detected edges. Here, some edges connect two different classification types of events, and the major types are System and Management. These two types are related to external operative connections, which cause login events (in Management) and UI process starting events (in System). This is not contrary to our intuition that remote login events appear more frequently than other events, because the login events usually follow the events of management, configuration changes, and troubles to be solved. The types related to network functions, like Network, Interface, VPN and Routing, are also major. These events usually affect multiple interfaces, paths or devices, causing more bases to be provided for the statistical decision. Our causal analysis is beneficial for analyzing these network functions.

In addition, we show the number of neighboring nodes of edges between nodes belonging to the same type. Most of edges in Routing-EGP (i.e., BGP events) and VPN (i.e., MPLS events) are included inside their 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 System and Management are also included inside their own type (i.e., inner-types). This means the events of external operative connections, which form most detected edges, are independent from events of other types.

In contrast, most edges in Network are connected to other types (i.e., inner-types). Table VI shows the matrix of event type combinations of the detected edges. We see that Network events appear mainly with events of Interface type. This is because network processes are related to interface state changes. For example, a network interface error causes network functions to fail. Some other combinations of event types, like System & Management, System & Service, and Network & Routing, are also major. This is because the combination of System & Management is related to external operative connections. The combination of System & Service is mainly the connection of NTP errors and message repetition alerts. The combination of Network & Routing indicates the BGP connecting sessions and their communications. Thus, these edges uncover the fundamental behaviors of the devices.

B. Postprocessing

As described in § IV-D, some detected edges contain no information regarding the importance of the causality that may help network operators. These edges appear redundantly in many different DAGs (i.e., datasets of different days), which prevent operators from finding essential information. For example, we find more than 10 edges between remote login events and user interface (UI) startup events every day in this dataset. These are regular behaviors of the network devices, and do not usually interest operators. In the spatio-temporal viewpoint, the edges of the same meanings can appear frequently for two reasons: one is on different days and the other is on different pairs of network devices. If some edges appear much more frequently for these reasons, such edges are less important for troubleshooting because they show the regular behaviors of the system.

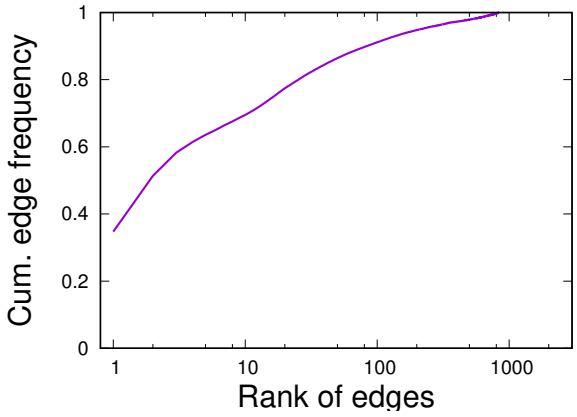


Fig. 12. Rank of edges and their number of appearances

To evaluate the effectiveness of our postprocessing method to decrease redundant edges, we investigate the distribution of detected edges of the same meanings (i.e., edges between nodes of events with the same log templates). Figure 12 represents the cumulative sum of such edges detected in all datasets. The PC algorithm produces 16,196 edges (= 35.5 edges/day). The top 5% of edges (42 out of 843 different edges) accounted for 85% of edge appearances. These edges appear in more than 35 DAGs, which is frequent enough for operators to consider them as regular behaviors. By manually observing the appearing edges, we find that most of them show causality between an input command and its acknowledgment. Thus, we filter them (with a threshold; 5%) and concentrate on the rest of them (2,438 edges = 5.3 edges/day). This postprocessing takes 20 seconds for all datasets.

Here, we take a look at the distribution of remaining edges after the postprocessing method, shown in Table V. Monitor events, consisting of SNMP messages, are likely to provide more meaningful information than other types. This is not contrary to our intuition because SNMP events appear with some unexpected behaviors of devices.

C. Case studies

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

1) Login failure: The first case is related to login failure events. A Layer-2 switch A is periodically logged in from another device X by a monitoring script. In some periods, device A reported login failure events. During these periods, we find BGP events in router B, a gateway of A, which causes a loss of connection. These BGP events are related to two ASes to which device A belongs. A blackout was actually recorded in the trouble ticket on the same day. This blackout event affects the devices related to the two ASes. We can estimate that this blackout event causes an interface error and a BGP connection initialization and finally causes login failures.

The log templates detected by the proposed system are shown in Figure 13. The first two templates (Template ID: 55, 56) are a part of the BGP connection initialization. The third one (ID: 58) is a failure of remote login. The fourth one (ID: 107) is related to the restoration of network interfaces,

TABLE VI
COMBINATIONS OF TYPES OF EVENTS THAT FORM CAUSAL EDGES. THIS TABLE DOES NOT CONSIDER THE DIRECTIONS OF EDGES.

Type	System	Network	Interface	Service	Mgmt	Monitor	VPN	Rt-EGP	Rt-IGP
System	6,278	90	91	51	1,003	47	4	32	0
Network	90	250	309	5	36	42	21	76	2
Interface	91	309	801	13	86	9	0	27	0
Service	51	5	13	67	23	20	0	15	0
Mgmt	1,003	36	86	23	5,004	7	0	25	0
Monitor	47	42	9	20	7	97	13	12	0
VPN	4	21	0	0	0	13	542	3	1
Rt-EGP	32	76	27	15	25	12	3	1,085	1
Rt-IGP	0	2	0	0	0	0	1	1	8

```
55[egp] (rpd[**]: bgp_hold_timeout:**: NOTIFICATION sent to
   ** (** AS **): code ** (Hold Timer Expired Error), Reason:...
56[egp] (rpd[**]: RPD_BGP_NEIGHBOR_STATE_CHANGED: BGP peer **
   (** AS **) changed state from ** to ** (event **))
58[mgmt] ([**]: EVT ** ** ** ACCESS ** **:** Login incorrect
   **)
107[interface] ([**]: EVT ** ** ** PORT ** ** **:** Port up.)
224[system] ([**]: RSP ** ** **(**):Can't execute.)
```

Fig. 13. Detected log templates (Login failure)

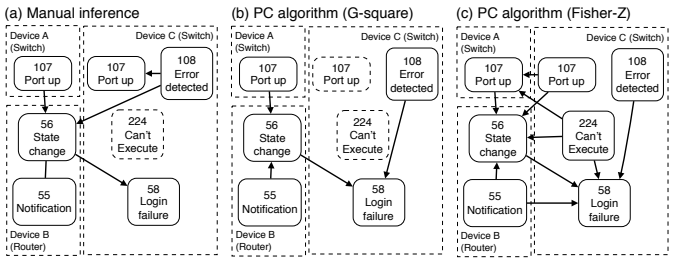


Fig. 14. Ground truth and detected causalities (Login failure)

which means the restart of devices after the blackout event. The other one (ID: 224) is a part of the manual operations, which is not directly related to this blackout. In our data, we observe two events of login failure and BGP events at the same time of login failure events for every two related ASes.

Figure 14 shows a plausible manual inference of this failure (a) and causality graphs generated by using the PC algorithm using the G-square test (b), and that using Fisher-Z test (c). In the figures, we manually added labels of events in the DAGs for readers to easily see their meanings. The DAG (b) is composed of four edges among the mentioned log templates, showing the causes of login failure events with allowable errors. Here, we successfully detect that the login failure events are caused by the BGP connection initializations. In this DAG, we can see some edges between a Rt-EGP event and that of other event types, like Interface and Management. These over-type edges are more helpful for operators than inner-type ones that is self-evident in many cases. According to Table VI, these combinations of event types are seen multiple times in whole dataset, which means similar helpful edges are detected.

Meanwhile, Fisher-Z test generates a more complicated DAG (c) that forms many triangles, but this also indicates the relation of BGP connection initialization and login failure event. In any case, our method provides useful information so as to easily understand the behavior of related devices.

```
55[egp] (rpd[**]: bgp_hold_timeout:**: NOTIFICATION sent to
   ** (** AS **): code ** (Hold Timer Expired Error), Reason:...
56[egp] (rpd[**]: RPD_BGP_NEIGHBOR_STATE_CHANGED: BGP peer **
   (** AS **) changed state from ** to ** (event **))
106[system] (mgd[**]: UI_CHILD_EXITED: Child exited: PID **,
   status **, command '**')
1406[system] (**)
1420[network] (** PFE[**]Aliveness Turning Destination ** on)
1423[interface] (** MIC(**)(**): SFP+ plugged in)
```

Fig. 15. Detected log templates (Hardware module alert)

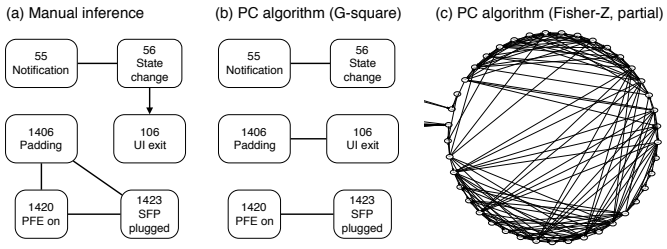


Fig. 16. Ground truth and detected causalities (Hardware module alert)

2) Hardware module alert: The second case is about physical interfaces on routers. A router is periodically monitored by a script. In some periods the monitoring command was abnormally terminated, we find a large number of alert messages about a corresponding physical interface. Also, some BGP connection initialization events appear before these events.

The detected log templates are shown in Figure 15. The first two templates (Template ID: 55, 56) are a part of the BGP connection initialization. These messages are reported once for four ASes. The third one (ID: 106) is reported when a child process called by a remote UI is abnormally terminated. (Instead, "UI_CHILD_STATUS" message is output in a normal situation.) This event appears for several times due to the retry. The other three templates (Template ID: 1406, 1420, 1423) show a part of the startup processes of the hardware module. The fourth one (ID:1406) looks strange as it seems not to show any information in its log template, but the variable is actually equivalent to a fixed word that points to a hardware module mentioned by others (1420 and 1423).

The BGP events are seen about one minute before the process termination events and the hardware module events. We can estimate that the hardware module was abnormally stopped, causing the initialization of BGP connections, and then began the starting processes. This trouble was actually

```
107[interface] ([**]: EVT ** ** ** PORT ** ** **:** Port up.)
108[interface] ([**]: EVT ** ** ** PORT ** ** **:** Error
   detected on the port.)
56[egp] (rpd[**]: RPD_BGP_NEIGHBOR_STATE_CHANGED: BGP peer **
   (** AS **) changed state from ** to ** (event **))
55[egp] (rpd[**]: bgp_hold_timeout:**: NOTIFICATION sent to **
   (** AS **): code ** (Hold Timer Expired Error), Reason:...
43[egp] (rpd[**]: bgp_send: sending ** bytes to ** (** AS **)
   blocked (no spooling requested): Resource temporarily
   unavailable)
5[monitor] (last message repeated ** times)
328[egp] (rpd[**]: ** (** AS **): reseting pending active
   connection)
```

Fig. 17. Detected log templates (BGP)

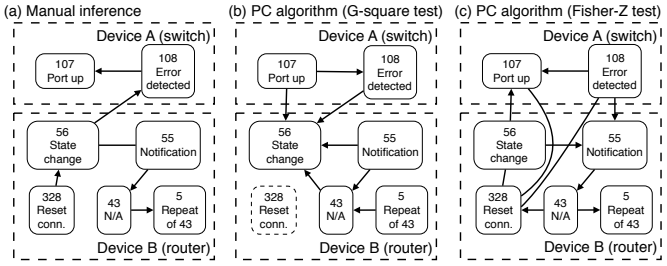


Fig. 18. Ground truth and detected causalities (BGP)

reported in a trouble ticket.

Figure 16 shows a plausible ground truth of this failure (a), DAG generated by using the PC algorithm using the G-square test (b), and a part of that using the Fisher-Z test (c). Our proposed method detects three related edges in (b): one is between the two BGP events, and the others are among process termination events and hardware module events. These edges are slightly different from those of our ground truth. It is difficult to generate DAGs that an operator expects only on the basis of their appearance time-series. Still, the information is helpful for the network operators to understand what happens in the system. In contrast, Fisher-Z forms a connected DAG of 70 events with 270 edges. This DAG consists of many triangle edges, and is too complex for operators to understand its behavior. This DAG is an example showing Fisher-Z leaves more false positive edges (discussed in § VI-B).

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

Figure 17 indicates the detected log templates in this failure. There are two log templates for one router, and five for the other. IDs 107 and 108 show the network interface error. ID 56 is a report of a BGP state change at the counterpart caused by the network error. ID 55 is a process of resetting a BGP connection, which repeats with some unknown causes. IDs 43 and 5 are events derived from a BGP connection resetting, which appear very differently from others. ID 328 shows a BGP connection resetting failure, which appears sporadically on BGP connection resetting process. The root cause of the connection resetting does not appear in the dataset.

Figure 18 shows a plausible ground truth of this failure (a) and causality graphs generated by using the PC algorithm (b), and (c). We again find that the result of our method with the G-square test is close to the ground truth. Despite some errors of directions of edges, the DAG helps network operators to understand the behaviors of devices. Although Fisher-Z also generate a DAG similar to the ground truth, the DAG has some redundant edges that connect events of indirect causality.

In summary, we show three cases of trouble with the detected DAGs. Two of them are recorded in trouble tickets, and we observed similar types of trouble in the tickets (see § VII-D). In addition, we found multiple troubles that are not recorded in tickets but important for operations like § VII-C3 with the PC algorithm. Thus, our method is useful for various practical situations of troubleshooting.

TABLE VII
THE NUMBER OF TROUBLE TICKETS THAT IS ASSOCIATED WITH DETECTED EDGES IN OUR ALGORITHM

Event type	Associated tickets	All tickets	Detection rate
Rt-EGP	91	106	86%
System	11	36	31%
VPN	19	19	100%
Interface	10	15	67%
Monitor	7	10	70%
Network	1	1	100%
Mgmt	0	1	0%
Total	139	188	74%

D. Comparison with trouble tickets

We compare the detected events by the proposed algorithm to the 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 227 failure events in a year. The number of the syslog messages is 28, 194, 935 in this period. We do not find any raw log messages regarding for 39 failures out of 227 failures.

Here, we further investigate the 188 failures regarding messages. Table VII shows the number of tickets for which our proposed algorithm provides related information for the tickets. This table is divided by the event types that are the most related to each tickets. There are some major types in tickets, like Routing and System. These types correspond to communication failures and hardware errors.

The PC algorithm successfully detected the related information for some types, like Routing and VPN. These events likely appear multiple times because they record log messages in multiple steps and their failure can affect multiple paths. Such events give enough information sources to estimate the relations for the PC algorithm. For example, the trouble mentioned in § VII-C1 is related to a ticket of Routing. There are 12 messages in 6 different events clearly related to the trouble. In the generated DAG, four edges are detected among five of the six related events. In this case, our method extracts the most of the information in the log data.

On the other hand, less information is provided for System. The events of System are likely to appear only once or a few times. Their causality is difficult to determine statistically with such events. § VII-C2 is a minor case in which some edges are detected for the ticket of System. There are more than 1,000

alerts in a short time related to this trouble. We detected only a part of its behavior.

In total, 74% of tickets can be associated with some detected edges. Thus, our proposed method reasonably managed to detect large events recorded in the trouble tickets, when the related log messages are successfully recorded.

VIII. Discussion

We show the effectiveness of our proposed approach to detect the causality of failure events. The PC algorithm with G-square detects more effective DAGs than that with Fisher-Z in the troubleshooting in the case studies. Also, the conventional correlation-based methods have a problem that a large number of false relations hide specifically important information. In contrast, the PC algorithm successfully reduces false positives from detected edges, which also reduces operators' tasks for troubleshooting. Thus, we can say that our proposed method contributes to improve network system management.

We observe a case when the edge direction is different from our intuition in § VII-C. The PC algorithm determines edge directions only with the rules derived from the definition of DAG, so our method cannot determine directions of all edges. For obtaining more appropriate results, we should additionally use timestamp information in the original data. In this case, we should carefully treat two events from different devices because of communication latency and time synchronization.

We also reveal that G-square processes conditional independence tests faster than Fisher-Z in log data. Fisher-Z takes more than one day to process one-day-long time-series in some cases (7% of datasets), which impairs continuous availability in real-time operations. Some existing works [18], [21] usually use statistical methods based on Pearson correlation as well as Fisher-Z test. These existing studies also have a potential problem in terms of processing time and usually aim to diagnose related events of particular trouble. Unlike the existing work, our method based on G-square contributes to enable not only diagnosing but also mining causal relations to construct a knowledge base for further analysis.

Preprocessing before applying the PC algorithm is an important step for obtaining appropriate results. According to § V, 93% of the original messages are inadequate for the analysis because of their periodicity. We previously [1] used a preprocessing technique based on an auto-correlation coefficient with lag. We found that this technique has three problems. First, the technique requires candidates of fixed periodicity intervals. Second, it removes all outliers in periodic time-series that would be important for troubleshooting. Third, it cannot remove periodic events with non-fixed intervals with noise. We found that our new method definitely removes periodic events but leaves events that are more important for system operators. Similarly, we divided original messages into temporal-spatial subsets to avoid detecting false causal relations. In fact, we find that there are no reported network-wide failures even in a large disaster case [39]. Similarly, the recent literature points out that failures are isolated in time and space for many cases in a global scale network [2].

Postprocessing is also a crucial step to pinpoint the target events because the PC algorithm detects causality but does not provide any importance on network operation. Our study relies on the simple heuristics to remove frequently appearing edges. However, as causal inference and anomaly detection are orthogonal, we plan to combine the current heuristics with a more sophisticated method to highlight the important events.

Comparing the detected results to the trouble tickets, we find that our proposed method detects edges related to most of the tickets. The proposed algorithm especially detected edges of events involving communications to other devices. These events incur much cost for system operators to locate the root causes if they fail. Thus, we may say that our proposed method is helpful in improving the daily network operation. In fact, network operators cannot easily identify the cause of login failures shown in the case studies.

In practical situations, operators require failure information as soon as possible. The current methods are not the best for real-time processing due to a lack of incremental updates of the PC algorithm. In our experiment, the average processing time for generating DAGs is 50 seconds in Figure 7. However, the processing time depends on the complexity of a dataset. In other words, it takes longer, over 600 seconds, if some trouble happens that causes various alerts. For real-time processing, we need a combination our method with other methods such as fault localization techniques.

IX. Conclusion

In this paper, we propose a method to mine causality of network events from the large amount of heterogeneous network log messages. The key idea in this paper is the leveraging of the PC algorithm that reconstructs causal structures from a set of time series of events. We find that the PC algorithm can output more appropriate events by removing pseudo correlations by conditional independence. Furthermore, our pre- and post-processing steps are helpful in providing a small set of important events that have causality to the network operators. Through three case studies and a comparison with trouble ticket data in the paper, we highlight the effectiveness of the proposed algorithm using 15 months' worth of syslog data obtained from a Japanese research and educational network. For a part of our future work, we plan to further evaluate our method with other network data such as SINET5 [40].

Acknowledgment: 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] S. Kobayashi, K. Fukuda, and H. Esaki, "Mining causes of network events in log data with causal inference," in *Proc. of IEEE IM'17*, 2017, pp. 45–53.

[2] 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 *Proc. of ACM SIGCOMM'16*, 2016, pp. 58–72.

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

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

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

[6] S. Urushidani and et al., "Highly available network design and resource management of sinet4," *Telecomm. Systems*, vol. 56, pp. 33–47, 2014.

[7] T. Kimura, A. Watanabe, T. Toyono, and K. Ishibashi, "Proactive Failure Detection Learning Generation Patterns of Large-scale Network Logs," in *IEEE CNSM '11*, 2015, pp. 8–14.

[8] E. Chuah, S.-h. Kuo, P. Hiew, W.-c. Tjhi, G. Lee, J. Hammond, M. T. Michalewicz, T. Hung, and J. C. Browne, "Diagnosing the Root-Causes of Failures from Cluster Log Files," in *Proc. of IEEE HiPC'10*, 2010.

[9] A. Ambre and N. Shekokar, "Insider Threat Detection Using Log Analysis and Event Correlation," *Procedia Computer Science*, vol. 45, pp. 436–445, 2015.

[10] Z. Li and A. Oprea, "Operational security log analytics for enterprise breach detection," in *Proc. of IEEE SecDev'16*, no. 1, 2016, pp. 15–22.

[11] F. Salfner and M. Malek, "Using hidden semi-Markov models for effective online failure prediction," in *Proc. of IEEE SRDS'07*, 2007, pp. 161–174.

[12] K. Yamanishi and Y. Maruyama, "Dynamic syslog mining for network failure monitoring," in *Proc. of ACM KDD'05*, 2005, p. 499.

[13] I. Beschastnikh, Y. Brun, M. D. Ernst, and A. Krishnamurthy, "Inferring models of concurrent systems from logs of their behavior with CSight," in *Proc. of ICSE'14*, 2014, pp. 468–479.

[14] Q. Fu, J.-G. Lou, Y. Wang, and J. Li, "Execution anomaly detection in distributed systems through unstructured log analysis," in *IEEE ICDM' 09*, 2009, pp. 149–158.

[15] 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 *Proc. of IEEE INFOCOM'14*, 2014, pp. 610–618.

[16] R. Sipos, D. Fradkin, F. Moerchen, and Z. Wang, "Log-based predictive maintenance," in *Proc. of ACM KDD'14*, 2014, pp. 1867–1876.

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

[18] Z. Zheng, L. Yu, Z. Lan, and T. Jones, "3-Dimensional root cause diagnosis via co-analysis," in *Proc. of ICAC'12*, 2012, p. 181.

[19] K. Nagaraj, C. Killian, and J. Neville, "Structured Comparative Analysis of Systems Logs to Diagnose Performance Problems," in *Proc. NSDI'12*, 2012, pp. 1–14.

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

[21] 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 *Proc. of ACM SIGCOMM'09*, 2009, pp. 231–242.

[22] B. C. Tak, S. Tao, L. Yang, C. Zhu, and Y. Ruan, "LOGAN: Problem Diagnosis in the Cloud Using Log-Based Reference Models," in *Proc. of IEEE IC2E'16*, 2016, pp. 62–67.

[23] 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.

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

[25] 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 *Proc. of ACM SIGCOMM'14*, 2014, pp. 395–406.

[26] C. Scott, A. Panda, A. Krishnamurthy, V. Brajkovic, G. Necula, and S. Shenker, "Minimizing Faulty Executions of Distributed Systems," in *Preceedings of NSDI'16*, 2016, pp. 291–309.

[27] 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'14*, 2014, pp. 1887–1895.

[28] T. Verma and P. Judea, "An algorithm for deciding if a set of observed independencies has a causal explanation," in *Proc. of UAI'92*, 1992, pp. 323–330.

[29] D. Colombo and M. H. Maathuis, "Order-independent constraint-based causal structure learning," *J. Machine Learning Research*, vol. 15, no. 1, pp. 3741–3782, 2014.

[30] T. Le, T. Hoang, J. Li, L. Liu, H. Liu, and S. Hu, "A fast PC algorithm for high dimensional causal discovery with multi-core PCs," *IEEE/ACM Trans. Comp. Biology and Bioinformatics*, vol. 13, no. 9, pp. 1–13, 2014.

[31] J. Abellán, M. Gómez-Olmedo, and S. Moral, "Some variations on the PC algorithm," in *Proc. of PGM' 06*, 2006, pp. 1–8.

[32] R. Vaarandi, "A data clustering algorithm for mining patterns from event logs," in *Proc. of IEEE IPOM'03*, 2003, pp. 119–126.

[33] A. Makanju, A. N. Zincir-Heywood, and E. E. Milios, "Clustering event logs using iterative partitioning," in *Proc. of ACM KDD'09*, 2009, pp. 1255–1264.

[34] M. Mizutani, "Incremental mining of system log format," in *Proc. of IEEE SCC'13*, 2013, pp. 595–602.

[35] S. Kobayashi, K. Fukuda, and H. Esaki, "Towards an NLP-based log template generation algorithm for system log analysis," in *Proc. of CFI'14*, 2014, pp. 1–4.

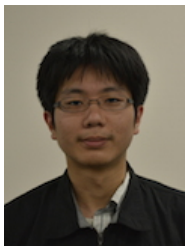
[36] R. E. Neapolitan, *Learning Bayesian Networks*. Prentice Hall, 2004.

[37] D. J. . Watts and S. Strogatz, "Collective dynamics of small-world networks," *Nature*, vol. 393, pp. 440–442, 1998.

[38] N. Harris and M. Drton, "PC algorithm for nonparanormal graphical models," *J. Machine Learning Research*, vol. 14, pp. 3365–3383, 2013.

[39] K. Fukuda and et al., "Impact of Tohoku Earthquake on R&E Network in Japan," in *Proc. of ACM CoNEXT WoID*, 2011, p. 6.

[40] T. Kurimoto and et al., "SINET5: A low-latency and high-bandwidth backbone network for SDN/NFV Era," in *Proc. of IEEE ICC'17*, 2017.



Satoru Kobayashi received the M.S. degree in information science and technology from the University of Tokyo, Tokyo Japan, in 2015. He is a Ph.D candidate at the Graduate school of Information Science and Technology, the University of Tokyo. His research interests are network management and data mining.

Kazuki Otomo is a Master's student at the Graduate school of Information Science and Technology, the University of Tokyo. His research interest includes knowledge extraction in network time series.

Kensuke Fukuda received the Ph.D. degree in computer science from Keio University, Kanagawa Japan, in 1999.

He is an Associate Professor with the National Institute of Informatics (NII) and the Graduate University for Advanced Studies (SOKENDAI). His research interests span Internet traffic analysis, anomaly detection, modeling networks and QoS over the Internet.

Hiroshi Esaki received the Ph.D. degree in electronic engineering from the University of Tokyo, Tokyo, Japan, in 1998.

He is a Professor at the Graduate school of Information Science and Technology, the University of Tokyo. Currently, he is Vice president of JPNIC, the Director of the WIDE Project, and the Board of Trustees of the Internet Society.